Nanyang Technological University

Computer-Assisted Inbetween Generation for Cel Animation

A Thesis
Submitted to the School of Computer Engineering
of the Nanyang Technological University

by

Chen Quan

for the Degree of Doctor of Philosophy

Feb 22, 2008

Abstract

Inbetween drawing is one of the most time-consuming and labor-intensive procedures in animation production. Much research has been done on automatic inbetween generation but it remains a tough issue. In image-based area, one of the difficulties lies in correspondence between keyframes. Many existing methods apply pixel-level matching without considering structure or motion of characters. This consequently results in incorrect matching between keyframes and hence unsmooth inbetweens. In vector-based area, common interpolation methods purely rely on mathematical functions of vector representation. This often leads to distorted lines, shapes and motion.

In this thesis, we investigate a wide range of information in keyframe drawings and accordingly propose several techniques by extracting the useful information and employ it into the process of inbetween generation. Our research is conducted in both image-based and vector-based areas.

In image-based area, we aim to solve the problem of incorrect matching between keyframes. Inspired by traditional inbetween drawing practices, we propose an approach to handle global and local motion of a character respectively. Keyframe relocation technique is proposed to minimize global motion and Modified Feature-based Matching Algorithm (MFBA) is adopted to estimate the motion of local changes. By integrating the two methods, accurate matching is achieved. Our approach is automatic and provides a robust solution for gray scale images as well as line drawings. For a character with complicated structure, we further propose a novel approach to segment the character into components based on auto-extracted skeleton and then estimate the motion of each component. This technique deals with the situation where various components of a character have different movements.

In vector-based area, we aim to solve different kinds of distortion problems resulting from linear interpolation. Firstly we propose a method to automatically extract feature

points of strokes in keyframes. By preserving correspondence of feature points in inbetweens, the shape of strokes is well preserved. Next we advance our proposed approach in image-based area, which handles global and local motion respectively, and apply it to vector representation. Hierarchical structure is established and kinematic inbetweening is applied, which makes the technique more robust for complicated characters. We further present a method to detect junction points between strokes in keyframes. By preserving junction points in inbetweens, the relationship between strokes is preserved as well.

Based on the requirement of converting scanned line drawings to vector representation, we finally propose a novel vectorization technique. It captures the variation of stroke width, which is a critical feature of original hand drawings. Our technique is automatic and efficient. It achieves high resemblance between original images and vectorized representations, and reduces data size significantly. It copes with different styles of drawing, such as Chinese drawing, Japanese animes and Hollywood animations. The technique stands as a firm base for further applications such as auto-painting and complete automation of inbetween generation in the future.

Extensive experiments have been carried out and the promising results prove effectiveness by incorporating various information to solve different problems in image-based and vector-based areas. Compared with common methods, our techniques improve the quality of inbetweens. Smoother and more realistic motion is generated. They are applicable to a wide range of situations of inbetweening. We believe our techniques can be used to assist animators and reduce time, labor and cost in animation production.

Acknowledgments

First of all, I would like to express my sincere appreciation and gratitude to Dr. Tian Feng, my PhD supervisor, and Prof. Seah Hock Soon for giving me the opportunity to explore and research into this interesting work and for their instruction, criticism and encouragement throughout the process. Their enthusiasm and rigor to research work greatly benefit me. The accomplishment of this thesis would not have been possible without their dedication, valuable knowledge and guidance.

I am grateful for the financial support of this research provided by Nanyang Technological University. I would also like to thank the staff in CGIT, CAMTech and IERC for technical and administrative support.

I would like to extend my appreciation to Dr. Wu Zhongke, Dr. Qian Kemao, Mr. Qiu Jie, Mr. Lv Ji, Mr. Lv Peng, Miss Xiao Xian, Mr. Lv Yixiang, Mr. Xie Xuexiang, Miss Li Li, Mr. Lai Kok Sen and Miss Zhang Haiwen, who have helped me and broadened my knowledge in the field through the period of research.

Last but not least, I would like to thank my family for their support, encouragement and blessings.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and Objectives

From magnificent Disney productions to popular Japanese animes, 2D animations, or cel animations, bring joy to children and adults alike. However, making traditional 2D animation is time-consuming and labor-intensive, which mainly consists of the following steps: story boarding, character design, drawing(keyframes/inbetweens), inking/painting and compositing [69].

To produce a smooth motion sequence, senior animators first draw *keyframes* depicting the extreme postures of motion. Assistant animators then draw *inbetweens* according to the keyframes [83]. Among various procedures, inbetween drawing is most tedious. Together with inking/painting, it takes up a large proportion, approximately 60%, of the total labor [88]. If inbetweens can be automatically generated by computer, or even partly, a huge amount of labor and time can be saved. It will significantly reduce production cost and improve animation quality. On the other hand, it is appropriate to let computer do the work since inbetween drawing requires less artistry. This research is hence motivated by a desire to provide automatic and effective inbetween generation techniques and tools which can reduce turnaround time and labor and free animators for more artistic and creative work.

Up till now, many animation systems as well as inbetween techniques have been proposed. However, some of them only handle simple transformation such as tweening an object along a path; some of them require heavy user-intervention e.g. selecting and corresponding elements in keyframes; and the quality of inbetween result does not meet the requirement of real production. Many of the problems are due to low-level matching and interpolation in inbetween generation lacking information of drawing contents. The objective of this research is to extract more information from keyframes and incorporate it into inbetween generation. We will investigate information such as pose, shapes, motion, and structure of characters as well as features and relationship of strokes in keyframes. Subsequently we will propose several techniques to extract different kinds of information and utilize it in inbetween generation to produce smoother and more realistic inbetween results.

1.2 Approach Overview

Based on the requirement of practical animation production, our work is proceeded in two categories: raster-based and vector-based inbetween generation.

Nowadays, most traditional animators are still used to draw with pencil and paper. Our raster-based techniques handle raster images scanned into computer from animators' drawing on paper. We propose an approach to handle global and local motion of a character respectively. In our approach, Principle Component Analysis (PCA) [18], is applied to estimate the respective pose of the character in keyframes i.e. the location, the orientation and the magnitude. Global motion can thus be computed and minimized by relocating the keyframes. Modified Feature-based Matching Algorithm (MFBA) is adopted to handle local deformation. After the inbetweens of the relocated keyframes are generated, the pose is restored by interpolating the global motion to produce final inbetweens.

To handle more complicated situations, we propose a novel technique based on segmentation. A character is segmented into approximately rigid components based on the skeleton extracted by SUSAN thinning algorithm [51]. Using a graph matching scheme, the segmented components of the character are corresponded in different keyframes. Global motion minimization is applied to each component to generate inbetweens. The interpolated components are finally composited to produce final inbetweens.

On the other hand, with more computerization of animation production, it is a trend to produce animations in vector form. Animators are making the shift from traditional hand drawing to computer assisted creation. In our research in vector-based area, several techniques are proposed to tackle different problems in inbetween generation of vector drawings. Firstly we propose an approach of feature-based stroke interpolation. Feature points depicting high curvature on strokes are extracted. Interpolation is then performed based on the correspondence of feature points on strokes in corresponding keyframes, which captures the characteristics of original drawings and preserves shape of strokes. Next we propose a novel technique of motion-enhanced inbetween generation for vector drawings. By adapting the global motion minimization approach proposed in our raster-based work, hierarchical global motion of characters is tackled, which eliminates shape distortion in rotation. Finally we introduce a method to detect junction points between strokes in keyframes and preserve them through inbetweens.

There is also a requirement of vectorizing scanned hand drawings to facilitate further processing such as editing and coloring. We hence propose a novel vectorization technique to convert raster line drawings to vector representations. To preserve the width variance of strokes, data disks along strokes are sampled. After being thinned one-pixel wide, the raster image is traced and divided into segments. Sample points are adjusted to obtain center points along the segments, while radii are computed at the same time. Feature points are detected to better preserve shape of strokes after vectorization.

1.3 Thesis Organization

This chapter explains the motivation, the objectives and the overview of our research. Chapter 2 reviews 2D animation production process, commercial 2D animation systems, and existing animation techniques by other researchers for keyframe creation and inbetween generation. In the ensuing chapters, our research work on computer-assisted inbetween generation for 2D animation is explained in detail. In Chapter 3, We elucidate our proposed approaches of inbetween generation for raster images. Modified Feature-based Matching Algorithm (*MFBA*), global motion minimization and skeleton-based segmentation will be explained in detail. In Chapter 4, we introduce our inbetween generation techniques based on the vector representation by disk B-spline curve (*DBSC*). Our proposed techniques of feature-based stroke interpolation, motion-enhanced inbetween generation and junction preservation in inbetween generation are described in turn. A novel approach of vectorizing raster line drawings is presented in Chapter 5. Conclusions and discussions on future work are made in the last chapter.

Chapter 2

Survey

2.1 Animation Production Process

It is necessary and important to have an overview of the workflow in traditional animation production and to understand how traditional animators conventionally do their work before developing approaches that will really meet the needs. In the following parts of this section, the conventional steps of 2D animation production are explained, among which keyframe and inbetween drawing process is described in detail.

2.1.1 Traditional Animation

Traditional animation is also referred to as cel animation, or hand-drawn animation. In a traditional animation, each frame is drawn by hand. The major steps to make a traditional animation [53, 83] are illustrated in Figure 2.1 and will be described as follows. For higher efficiency, there is often an overlap in the above procedures and even parallelism in practice [83].

Storyboard

To depict the story which the animation is to tell, *synopsis* and *scenario* play the role of a summary and a detailed text respectively. The *storyboard* offers a visual description

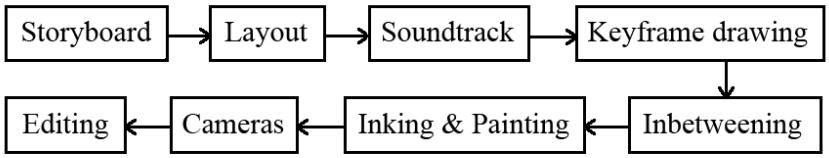


Figure 2.1: Traditional animation production process

of the film's key moments, composed of a sequence of drawings that define specific actions.

Layout

Based on the storyboard, this step mainly consists of character designing and action plotting. It is important to illustrate the appearance of individual character in different poses and from various perspectives as well as the proportion between each character. This will serve as a reference when the entire task of drawing is later distributed to a team of animators so that the drawings from different animators remain consistent.

Soundtrack

In conventional animation, soundtrack is usually recorded preceding the animating process. Motion must be analyzed frame by frame in order to achieve synchronization with dialog and music. Alternatively, scoring and sound effects can be added in final editing and mixing.

Keyframe drawing

The drawing of frames is the spirit of animation. In this process, animator draws sequences of animation on sheets of paper perforated to fit the peg bars on the *light box*. A light box, also called the *trace box*, as shown in Figure 2.2, holds a fluorescent light fixture that shines up through a rotatable disk of frosted glass or milky-white plastic. This allows animators to see through the sheets and easily trace images on a sheet of

paper below the one he is working on. The peg bar on the disk helps to register the sheets with corresponding holes.

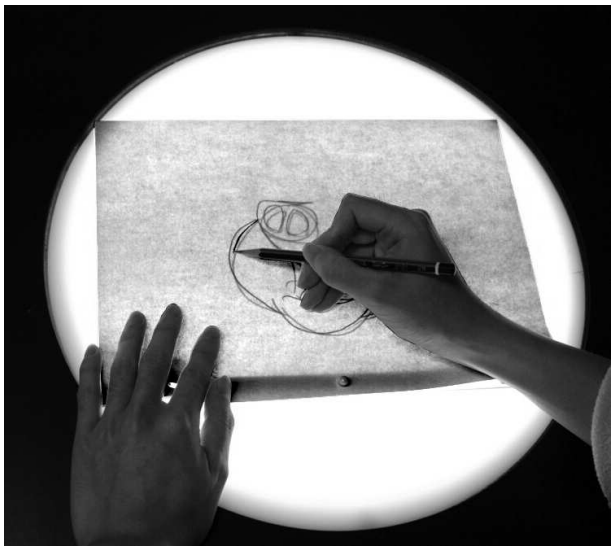
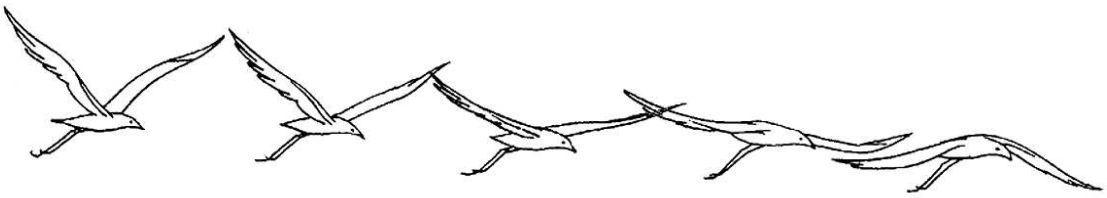


Figure 2.2: Drawing frames over light box

A *keyframe animator* will draw *keyframes* in a scene, using the character layouts as a guide. They draw enough frames to get across major points of the action. A frame often comprises multiple layers. Different characters and backgrounds are usually drawn in different layers so that characters may move without disturbing backgrounds, or backgrounds may vary by scrolling to one side as characters move. Only what moves in each frame must be drawn anew. All the layers will be composited into a single frame. During this stage as well as the subsequent inbetween drawing process, line testing will be carried out, where the pencil drawings are quickly photographed or scanned and synced with necessary soundtracks, in order to get a preview of the actions and rectify flaws. An animator may be required to re-work on a scene many times before the director approves it.

Inbetweening

Once the keyframes are finished, the drawings go through a clean-up process and are forwarded to the *inbetweeners*. The inbetweeners will fill up remaining frames which are placed between keyframes in order to make the movement look smooth. These frames are called *inbetweens*. Inbetweeners create inbetweens according to the changes between keyframes. To draw an inbetween frame, the given two keyframes and a blank sheet for inbetween are first registered by holes onto the pegs of the light box for alignment. The pegs maintain registration so that all the keyframes and the inbetween are in exact relationship to each other. The light shines from the back of the sheets making the sheets look transparent so that the drawing on all the sheets can be visualized. The inbetweener examines the change between the keyframes and draws the mid-elements of corresponding points, lines and shapes based on which he draws inbetweens. Once finished, starting from the first keyframe, every frame in the sequence changes slightly from the previous one until finally the second keyframe is reached. An example is illustrated in Figure 2.3. The resulting drawings are again pencil-tested for approval.



Keyframe 1 Inbetween 1 Inbetween 2 Inbetween 3 Keyframe 2

Figure 2.3: Keyframes and inbetweens

Among various procedures of traditional animation production, inbetween drawing is one of the most tedious ones. A 90-minute cel animation consists of more than 100,000 frames, a large proportion of which are inbetweens. Automation of inbetween generation would allow artists to leave this tedious work to the computer and concentrate on the

more creative work of drawing the keyframes. However, quality is a crucial issue. As recapitulated by John Halas [35], one of the world's most famous animators, 'movement is the essence of animation'. Animators must make sure that objects move in a convincing manner. Inbetweens play an important part in the quality of the final animation. Despite the computerization of some other procedures in the production, so far inbetweens are mostly drawn manually. The results of some automatic inbetween generation systems do not reach the standard of the industry.

Inking and painting

In early years, the completed pencil drawings are subsequently transferred to acetate cels by xerography. The inking and painting stage is gone through on the cels with the right degree of opacity. Nowadays it is a common practice to scan the drawings into computer and paint them using a wide variety of software tools.

Cameras

After compositing all the layers of a frame and carefully checking the action in the scenes assuring that everything is properly executed and identified, all the frames are brought together on the rostrum camera, illuminated and photographed onto color films or videotapes frame by frame at certain rate, e.g. 24 frames per second.

Editing

To transform the shot film into a final product, postprocessing and editing is applied where the film is assembled, sorted and spliced to polish the final production.

2.1.2 Computer-Assisted Animation

With the development of computer techniques, the process of traditional 2D animation production gradually moves to the computer-assisted stage. Quite a few procedures are being carried out using computer animation systems nowadays. For example, inking and painting are usually done digitally after the outline drawings are scanned into the computer instead of being transferred to cels and then colored by hand. Compositing and editing is mainly done on computer as well. Using computers facilitates easier exchange of artwork and cooperation of production between departments, studios and even countries.

For frame drawing process, many animators still use pencil and paper. The drawings are scanned into computer and stored as raster images for further processing. On the other hand, it has also become possible for animators to directly draw frames in computer using a digital tablet, as shown in Figure 2.4. It is similar to the way of drawing on papers but easier for modification. The drawings can be stored as either raster images or vector form.

Raster images capture the exact styles and characteristics an animator desires and it is easier and more intuitive for a traditionally-trained animator to control the appearance of the final work. Artists can produce higher-quality drawings using their paper drawing skills, free from possible limitations imposed by vector drawing tools. The process is more straight-forward. However, the higher quality is demanded which requires larger resolution of images, the more storage space is needed for the image files.

Vector drawings are easier and faster for modification. They have advantages, e.g. scalability and compactness. They can be scaled and rotated without losing fidelity, which allows easy reuse by simply changing output resolution. Vector form is concise in representation hence is of small storage size. It is more suitable than raster drawings for animation production to be displayed on high definition display devices, which are becoming more popular and may soon be dominant in market.

Some techniques are also developed to convert raster drawings to vector form which further facilitates the drawing process. It is likely that paperless 2D animation will supersede the traditional pencil-and-paper animation, not too far in the future.

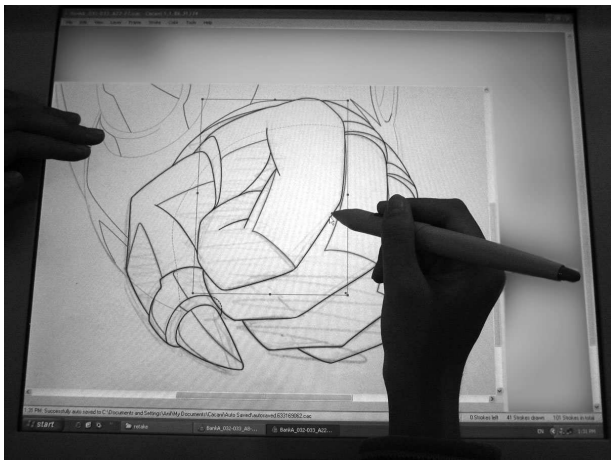


Figure 2.4: Drawing frames on a tablet PC

No matter whether an animator uses raster or vector drawing, it is difficult to generate inbetweens automatically. In most cases, inbetweens are still drawn frame by frame manually. Therefore it is one of our major goals to make automatic inbetween generation easier and more accurate.

2.2 Commercial 2D Animation Systems

There are quite a few commercial 2D animation systems available to assist animators in the production. We will go through these systems, concentrating on the modules or techniques for frame drawing and inbetween generation, if any.

As depicted in [42], the two main classes of automated inbetweening systems are based on templates [58, 4] or explicit correspondence [86]. In template-based systems, a designer creates a template for each character. Animators must then restrict the character's movements according to the template. The correspondence between each keyframe and

the template must be specified. Explicit correspondence systems use two keyframes and generate inbetweens on a curve-by-curve basis. An operator is also required to specify the correspondence between the curves in the two keyframes. Sometimes, a zero-length curve must be created when a new stroke appears or disappears between keyframes.

[**Tic Tac Toon**] [42]

TicTacToon is a system for professional 2D animation studios produced by Toon Boom Technologies Inc. It is the first animation system which uses vector-based sketching and painting that replaces the traditional paper-based production process. It provides interface enabling professionals to sketch and draw like they do on paper, and transforms a pen trajectory with varying pressure into a stroke of varying thickness using Bézier curves and least squares curve fitting, in realtime. However, it does not provide a computer-assisted inbetweening module. Animators need to draw inbetweens frame by frame according to keyframes as they do in traditional way. TicTacToon has been discontinued by Toon Boom Technologies Inc.

[**Toon Boom Studio**] [8]

Toon Boom Technologies Inc. later developed a sequence of animation software packages, including Toon Boom Studio for individuals and Toon Boom Harmony, an enterprise version for studios, advanced from their previous product 'USAnimation'. With integrated workflow and configurable user interface, their systems cover most of the tasks in animation production and are good for producing paper-cut animations. The products are vector-based, resolution-independent, supporting vectorization from scanned bitmaps or directly drawing on tablet. Harmony is armed with more powerful features, such as morphing, inverse kinematics and its glue effect. Inbetweens can be created using the morphing tools. However, it is object-based, i.e. user creates a vector-based object and

transforms it to create the next keyframe. The practice is limited to relatively simple movements. It is quite a different solution from traditional animation where animators draw keyframes separately. Here animators only create the first frame and alter it by some transformation to be the second keyframes. Then the system interpolates or morphs the transformation to generate inbetweens. It is not intuitive and sometimes troublesome and complicated to control and it can only generate limited motion.

[**Animo**] [4]

Animo is a software for professional animation production which integrates 2D and 3D animation. It is produced by Cambridge Animation Systems and was first launched in 1990. Animo uses a mixture of bitmap and vector technology to provide both clean, fast, animateable vector paint and the accuracy and essence of the original drawings though bitmaps. Animo uses the original scanned lines from animators' drawing to perform the subsequent processes such as painting, which is raster-based. Vector painting and drawing tools allow users to create inbetween frames but only camera and element positions can be keyframed and automatically inbetweened.

[**Retas!Pro**] [7]

RETAS! (Revolutionary Engineering Total Animation System) Pro is a digital animation production package produced by CELSYS, Inc. It is the leader software used in Japan. RETAS! is primarily a bitmap based system. One of its components, TraceMan, is the scanning application that allows users to scan their images into the system and trace the images so that the outlines are suitable for painting. It previously used conventional raster representation for traced strokes. In the recent version, it also supports vector tracing. After loading a raster image to be traced, user clicks on the lines to drop control points and the system automatically interpolates the control points into vector

representation. However, no inbetweening module is available.

[**Micromedia Flash**] [6]

Micromedia Flash is a well known vector animation software for web animation creation. It enables users to create vector-represented objects in keyframes. The parts of a character with different movement are defined as independent objects and manipulated separately. User creates keyframes by dragging the object to its starting and ending location in the keyframes. The moving path can also be specified. Flash will draw and transform the inbetween shapes.

Because of the limitations of the software and small file sizes for web distribution, a large part of animations created in Flash are simplistic limited animations, or in a cutout animation style. Some typical features of Flash animation could be: jerky natural movements (walks, gesture), lip movement without interpolation, abrupt changes from front to profile view or even no head turns at all. Generally Flash is used for quick and less complicated animation production which does not require high quality as traditional animation does.

[**Mirage Studio**] [3]

Mirage Studio is a paperless digital animation system designed by Bauhaus Software, Inc. It is a bitmap-based system, which digitally produces frames similar to those drawn on real paper. It uses bitmap to produce stylish brush effects, change opacity, and accurately simulate natural-media tools, such as crayon, charcoal, oil painting, chalk etc. It can apply motion along linear or spline paths similar to the tweening function in Flash.

Table 2.1 compares the advantages and disadvantages of the above-mentioned animation systems. There are some other animation softwares such as Plastic Animation

Paper [1], Toonz [9], CelAction2D [5], Animation Stand [2] etc., none of which provide an inbetween generation module. In all, most existing animation systems do not support inbetweening function. A few systems with inbetweening function change the nature of inbetweening and limit its complexity.

Table 2.1: Comparison between representative animation systems

Systems	**Advantages**	**Disadvantages**
Tic Tac Toon	Vector-based	No inbetweening module
Toon Boom Studio	Vector-based Morphing tools Inverse kinematics	Limited simple motion Complicated to control
Animo	Mixture of bitmap and vector technology	Inbetween of only camera and element position
Retas!Pro	Bitmap-based Vector tracing	No inbetweening module
Micromedia Flash	Vector-represented Object-based inbetweening	Simple limited animation
Mirage Studio	Bitmap-based Path-based motion	Simple tweening

2.3 Inbetween Generation Techniques

Catmull [23] presents an overview of computer-assisted animation problems and suggests possible solutions in several aspects to handle inbetweening. During the past decades, tremendous effort has been made concerning each proposed method. In this section, we will review previous research work on this topic, both in image-based and vector-based areas.

Image-based inbetween generation can be summarized as follows: given scanned keyframes of raster images, match pairs of drawings or match drawings to a template, i.e.

establish correspondence between components in the keyframes based on image properties, and interpolate inbetweens. Vector-based inbetweening techniques are similar but they handle input frames represented in vector form, usually by mathematical representation. There may not be distinct boundary between these two categories as in some methods there is a mixture of both image and vector techniques.

2.3.1 Curve Interpolation

Most inbetweening techniques are based on a curve-by-curve basis or an object-to-object basis [42]. Interpolation is performed on the corresponding elements in keyframes based on fitting a set of interpolating splines through key positions.

Some attempts are made to improve mechanical look, discontinuity and distortion of intermediate curves resulting from linear interpolation and target at smoother and more natural transition between target and source images. Reeves [79] presents an approach that gives animators more flexible control over interpolation through moving points which constrains both trajectory and dynamics of certain points in keyframes. Kochanek and Bartel [81] propose a method of using cubic interpolating splines. Three control parameters allow animators to change the tension, continuity, and bias of the splines. Each of the parameters can be used for either local or global control. Bartel and Hardock [65] study associations between spline curves within a frame that hold throughout inbetweens. Control points of curves are used as key points of interpolation to generate inbetweens.

These methods focus on interpolating one curve at a time without considering the correspondence problem of the numerous pairs of curves in two keyframes. In most of these methods, a one-to-one correspondence is assumed between components in keyframes and interpolation is performed to the corresponding elements. Alternatively, animators are required to manually specify explicit correspondences between curves in keyframes. Moreover, some parameters need to be manually controlled to obtain desired inbetween results. Therefore, it requires additional work of animators aside from drawing.

Kort [12] presents an algorithm for automatic mapping of the strokes in keyframes. Each keyframe is analyzed and classified into strokes, strokes chains and relations that hold among them. Correspondence is obtained according to the rules which specify changes allowed between relations. The correct matching of strokes is determined by a cost-function-based approach. Like some earlier attempts [76] for automatic stroke mapping, the approach works mostly for simple cases. When the drawing becomes more complicated, it frequently results in incorrect matching and requires indispensable manual correction.

2.3.2 Image Morphing

Image morphing, also referred to as *image metamorphosis*, is a powerful tool for visual effects in film and television and has always been a research interest. It enables smooth transition from a source image to a target one, in some way similar to inbetweening. A conventional method of generating an intermediate image is by cross-dissolving the source and the target images. This approach, however, cannot generate a high-quality image since the features of the objects in the source and the target images are generally not coincident with each other.

To assure smooth transformation, first the source and the target images are warped so that the features on both images are aligned to each other and then the colors of the warped images are cross-dissolved. First, animators need to establish the correspondence between two images with pairs of feature primitives, e.g., points, line segments or curves. Each primitive specifies an image feature. The feature correspondence is then used to compute mapping functions that define the spatial relationship between all points in both images. The mapping functions, or the warp functions, will be used to interpolate the positions of the features across the morph sequence. Once both images have been warped into alignment for intermediate feature positions, inbetween images are generated by intensity interpolation [30].

There are four main issues to overcome in morphing. First is to extract the features from the source and the target images, and to establish the correspondence between these features. Second is to interpolate these features to generate features for the intermediate images. Third is to define a warp function. And lastly is to control the transition rates of the shape and the color. These major problems influence the ease and effectiveness in generating high-quality metamorphosis sequences. Wolberg [30] gives a survey on various image morphing techniques addressing the problems, such as *radial basis functions* [57], *thin plate splines* [67, 61], *energy minimization* [68], and *multilevel free-form deformations* [68]. Other efforts are also made, such as [77, 60, 46].

In [46], an object-space morphing technique that blends the interiors of given two- or three-dimensional shapes in presented. Given a boundary vertex correspondence, the source and target shapes are decomposed into isomorphic simplicial complexes. The algorithm computes the compatible triangulation of the two shapes. Next, the optimal least-distorting morphing between each part of corresponding triangles is determined. The global transformation is defined as a transformation which minimizes the overall local deformation. This method as well as some others [22] handles image morphing using triangulation, which is not suitable for line drawings.

[33] presents solutions to the feature correspondence and feature interpolation problems in image morphing. User needs to specify the correspondence between the source and target images by drawing input curves on the features of the objects. The correspondence between these curves at the pixel level is computed by optimizing a cost function. Based on this correspondence, the input curves are approximated using Bézier curves. Then the Bézier curves and the connections among them are represented using a 'dependency graph'. Inbetweens are generated by interpolating the dependency graphs using the edge-angle blending techniques. The algorithm does not prevent the intersections between features during interpolation. The edge-angle blend technique for interpolating the dependency graph might distort the area of the intermediate shape.

2.3.3 Shape Blending

Shape blending is a research topic within the scope of image morphing but more specific on single-shape interpolation without considering intensity. Given two polygonal shapes, a continuous shape transformation from one to another is to be computed. It also consists of two problems: vertex correspondence and interpolation path.

A physical based approach is introduced in [84]. The given shapes are considered to be constructed of a piece of wire, and a solution is found whereby the first shape can be bent and/or stretched into the second one with a global minimum solution of a work function. The method requires users to specify the values of various parameters which control the intermediate shape change in order to achieve desired result. Due to the linear motion between corresponding vertices, some parts tend to distort especially when rotation occurs. Improvement is made in [86] by interpolating the intrinsic attributes of the shapes i.e. the edge lengths and the vertex angles, instead of the explicit coordinates of the polygon vertices. Some works have been presented to avoid self-intersection of the intermediate polygons. Shapira and Rappoport [50] use the star-skeleton representation for the polygons. Intermediate shapes are generated by blending the parametric description of these skeletons. Gotsman and Surazhsky [19] achieve the non-self-intersection property by blending compatible triangulations of corresponding point sets, which is performed by interpolating vertex barycentric coordinates instead of vertex locations. Surazhsky et. al. [80] propose a method for blending polygonal shapes with different topologies, which also guarantees non-self-intersections. Sederberg and Greenwood [85] make further improvements in shape blending by using B-spline curves. The shortcoming of these methods is that they only consider the interpolation between two object boundaries. Animation drawings are much more complicated. Matching shapes between two keyframes is in the first place difficult. It is also difficult to specify the parameters to

obtain desired results.

A common requirement of most approaches of image morphing or shape blending is a pre-specified initial correspondence between the source and the target images or shapes, which involves manual interference, more or less. Some algorithms such as [60] address the problem of automatic correspondence but are more suitable for the situations when the input images are sufficiently similar. Generally, user-interaction is inevitable without high-level understanding of the input images. This remains a research focus in the field. Besides, most of these methods do not specifically consider features or motion in inbteween generation, which may result in distortion of inbetweens.

2.3.4 Skeleton Technique

In order to have more control of complex motion dynamics in inbetweening, Burtnyk and Wein [58] introduce a high level of interaction utilizing skeleton drawings established in correspondence to a fully drawn character. This idea is derived from examination of the methods used in conventional animation where animators often sketch stick figure representations to draft keyframes. According to their method, a drawing is represented by a simple skeleton. The image is described in relative coordinates and the skeleton representation implies a central core of connected 'bones' with a surrounding image distribution. A distorted form of the image is extracted by modifying only the skeleton. Thus the problem is reduced to animating a stick figure representation of the image which will in turn impart the movement to the actual image sequence. Since the skeletons are simple images composed of only a few points, it is possible to provide a high level of interaction.

The result is impressive with smoother motion. However, this technique requires a huge amount of human interventions, not only in the skeleton representation setup but also in the manipulation of the skeleton for keyframe creation. The process of reconstructing the silhouette does not work well around joints and may require manual work.

Besides, the interpolation process is also based on stroke-to-stroke mapping, which requires an invariant order of stroke tracing from images, which is based on input order.

Owen and Willis [48] discuss a method of modelling cartoon characters that separates the appearance of the character from its movement. A shape tree and a skeleton are used respectively as representations. The main benefits are the ability to store and mix character movements, as defined by the changing position of the skeleton, and the separation of appearance from movement, allowing independent reuse of these two elements. However, heavy user-interference is required for establishing the structure. The technique is also object-based, similar to some animation systems such as [8]. We have explained earlier in Section 2.2 that the process of inbetween generation is different from traditional production.

2.3.5 2.5D Technique

In traditional animation, a frame often comprises multiple layers. For example, in order to make an object move in front of a background, animators use one layer for background and another for object so that multiple animators can work simultaneously on different layers. Each layer is drawn separately. Then all layers are composited together and transferred onto film by photographing. The concept of 2.5D is derived from this conventional practice. It maintains the drawing order of 2D objects so that the objects appear to be on hierarchies of parallel planes which compose the actual image. The elements on higher planes occlude elements on lower planes.

Litwinowicz [20] first brings the concept into his animation system, *Inkwell*, which attempts to provide animators a natural and intuitive way of drawing characters. The algorithm in [12] is also layer-based, assuming an invariant layering order.

Van Reeth [29] later suggests integrating 2.5D techiniques to support traditional animation. Together with Di Fiore, he introduces a method for automatic inbetweening

based on 2.5D modelling [28]. Characters and objects are modelled as sets of depth ordered primitives with respect to the x-axis and y-axis rotations. Every frame is depicted by a list indexed by two rotation angles. It holds the references to the underlying curve primitives as well as their drawing order in the frame. A set of extreme frames serving as a template is first set up. After specifying the rotation parameters of the keyframe, the intermediate frame is generated by interpolating the most similar extreme frames. A multi-level system architecture is also suggested. Level 0 starts with basic 2D drawing primitives (curves); level 1 comprises explicit 2.5D modelling structures; level 2 includes hierarchical information by means of skeletons and level 3 offers the opportunity to include high-level tools.

They further introduce a method of mimicing emotions of a character as seen from different viewpoints [27]. The concept of *Facial Emotion Channels* they proposed is similar to multi-layer technique. Every individual part of the face is called a *Facial Emotion Channel* and modelled independently. For each part, a set of versions depicting different emotions are modelled, which is controlled by user-input parameters. A created emotion from one point of view can be used as a reference to create that of another different point of view.

Using multi-layer simplifies the correspondence of keyframes by breaking a character into simple layered objects. However, it is rather troublesome thus unreasonable to require artists to draw every part of a character in different layers hence the method is applicable to a limited kind of movements i.e. animations of paper-cut-out style.

2.3.6 Image Matching

As discussed above, establishing correspondence between keyframes is an important task in inbetween generation. However, most aforementioned methods emphasize on the interpolation problem and the process of matching usually assumes pre-specification or requires user-specification.

Xie [55] tries to establish the correspondence of feature points in the images by comparing their characteristics such as curvature and distance between two neighboring feature points. Some experiments are made to apply the algorithm to handle affine transformation inbetween generation. Zeng and Yan [93] propose an approach to region feature extraction and region matching for cartoon image processing. By introducing the inertia coordinate system, the features of regions which is invariant to translation, rotation and scaling have been extracted. Fuzzy dissimilarity is used as a matching variable in the optimal matching. However, none of these methods handle big changes which are common in keyframes thus the applicability is limited.

Seah [72, 71] presents the *Modified Feature-Based Matching Algorithm*, MFBA, that achieves complete automation in inbetween generation without any user-intervention. It is derived from the *Feature-Based Matching Algorithm*, FBA [40], which treats the images as samples of a scene taken at a discrete time and determines the motion of all the pixels between two successive frames using multiple image feature correspondence. After matching keyframe pairs, a *Displacement Vector Field* (DVF) is generated. Then the DVF, comprising individual pixel displacement, is used to generate inbetweens. MFBA introduces various features as matching tokens and makes several modifications to the original FBA. However, this optical-flow-based algorithm is suitable for moderate changes in two keyframes. It may not match accurately when the changes become drastic, which limits the scope of application. In our research, we will investigate more information from the images and adopt MFBA as the foundation of image matching. More details will be given in Section 3.2.1.

2.3.7 Other Techniques

Most of the methods reviewed above lack 3D information thus can only handle limited circumstances, primarily in 2D. Some preliminary efforts have been made to incorporate 3D information to enhance inbetween generation.

In [26], approximate 3D models are employed to the production of cartoons. It aims to retain volumes, shapes, and proportions of the objects and ensures frame-to-frame coherence. Animators first sketch 2D circular and rounded forms as representations of objects. Free form stroke techniques are used to create and modify the 2D objects. Then the primitives are interpreted to automatically construct a 3D polygonal object of revolution. Modifying 3D objects and performing affine transformations upon them are also supported for creating new key poses without having to construct again. An object is manually oriented to a desired posture and animators draw the outlines according to the silhouette of the object as well as features and other details. Changing outlines in specific keyframes and adapting them to the feeling and effects is also allowed to help reach artistic goals. In this way, the approximate 3D models are used to guide animators throughout various stages of the animation process for retaining volumes.

Some other research focuses on employing full 3D input models, which are rendered and even animated in many different non-photorealistic styles. Rademacher [62] presents a view-dependent model wherein a 3D model changes shape based on the direction it is viewed from. The model consists of a base model and a description of the model's exact shape (key deformations) as seen from specific key viewpoints. Given an arbitrary viewpoint, the deformations are blended to generate a new, view-specific 3D model. However, animators still have to construct the base model and its deformations for each key viewpoint which is undoubtedly time-consuming.

Some recent methods use active contour to vectorize the input video sequences [16] or introduce 2.5D model [28] to generate inbetweens. Both methods require heavy user interaction. In [28], *non-photorealistic rendering* (NPR) techniques can generate possibly stylized cartoon using 3D geometrical model. However, the disadvantages are the needs to create complicated 3D models and difficulties to achieve lively movement.

Although our work focuses on 2D approaches and 3D is not within the scope of our research, it is a good direction to introduce 3D information in the future since it brings

more information and will help solve the problem of occlusion and 3D rotation.

In all, existing auto-inbetweening systems and techniques mainly have the following disadvantages. Crafting an animation with an automated inbetweening system could be slow. Many systems require limitation of input keyframes. Even with recent advances in algorithms, it remains difficult to match keyframe drawings without heavy user intervention. Moreover, many techniques require troublesome user-intervention to control the inbetweening process. Inputting and tuning simple inbetweens may take about the same amount of time as drawing them by hand, which is inefficient. The results of auto inbetweening are far from reaching the standard of real production. For purely 2D approaches, getting perspective right and retaining volumes and the overall shape of the objects are major problems. Most methods do not consider information from the drawings, such as structure or motion of the characters in keyframes.

2.4 Summary

In this chapter, we took an overview of traditional computer-assisted animation production. Both the image-based and the vector-based methods play an important role in industry nowadays and have their own advantages and disadvantages. Therefore it is necessary to conduct research in order to solve the problems in both areas. Next we made a survey of existing commercial 2D animation systems and various techniques of inbetween generation. Current results of computer generated inbetweens do not meet the standard of the real production. There are still many problems to solve both in keyframe matching and interpolation. Our proposed approaches, which will be introduced in the following chapters, aim to solve some of these problems. In image-based area, we aim to solve the correspondence problem between keyframes with little user intervention. In

vector-based area, we aim to solve different kinds of distortion problems resulting from linear interpolation so as to preserve shapes and characteristics of strokes.

Chapter 3

Inbetween Generation for Raster Images

3.1 Introduction

Despite rapid development of computer-assisted cartoon systems, it is difficult for many traditional animators to overcome their computer phobia. They are still used to drawing frames by pencil and paper and then scanning their drawings into computer for further processing like coloring and compositing. The techniques presented in this chapter aim to tackle this case: to automatically generate the inbetweens given keyframes of scanned raster images.

Image-based inbetween generation consists of two problems. One is to establish correspondence between keyframes and the other is to interpolate the corresponded elements to generate inbetweens. Many existing methods mentioned earlier in Section 2.3 only work on the interpolation problem, which assumes or requires manual initial correspondence between elements in the keyframes. Most methods which handle matching problem involve heavy user-intervention when motion of characters is big. Some automatic correspondence methods such as [55] only handle relatively similar keyframes. Therefore it is not easy to apply these methods in real production.

Examining the motion of a character, it can usually be divided into *global* movements

and *local* deformations. In conventional inbetweening process, when the overall translation or orientation of the character in the keyframes is big, it will also be quite difficult to directly draw the inbetweens. In this case, inbetweener adopts an approach of *relocation* to alleviate the drastic changes as shown in Figure 3.1. The two keyframes are not aligned using the pegs on the light box as usual. Instead, they are overlapped with certain excursion so that the two moments of the character on the sheets come to the same position. The register holes on top of the two sheets denote the transformation of the two sheets. Then a new sheet for inbetween drawing is placed onto the light box with its register holes in the midst of those of the keyframes, somehow like interpolating the two keyframe sheets. Now that the images on the two keyframes get closer, it is easier to draw the inbetweens. Similar method is applied for handling global 2D rotation. Sometimes inbetweeners must do this several times for different parts of the character. For example, if a character jumps and his arm also moves, inbetweener will first superimpose the body to draw its inbetweens and then superimpose the arm to work on its inbetweens.

Inspired by the traditional method, we propose a novel approach in this chapter to handle global and local motion respectively. We propose keyframe relocation to minimize global motion and adopt *Modified Feature-Based Matching Algorithm* (MFBA) to tackle local changes. For a complicated character, we propose a novel technique to segment it into components and apply motion estimation for each component. In the following sections, the details of the techniques will be elucidated.

3.2 Handling Local Deformation by MFBA

Modified Feature-Based Matching Algorithm, MFBA, is derived from the original *Feature-Based Matching Algorithm* (FBA) [40] and advanced by Seah [72, 71] for matching keyframe drawings and automatically generating inbetweens. The main point of MFBA

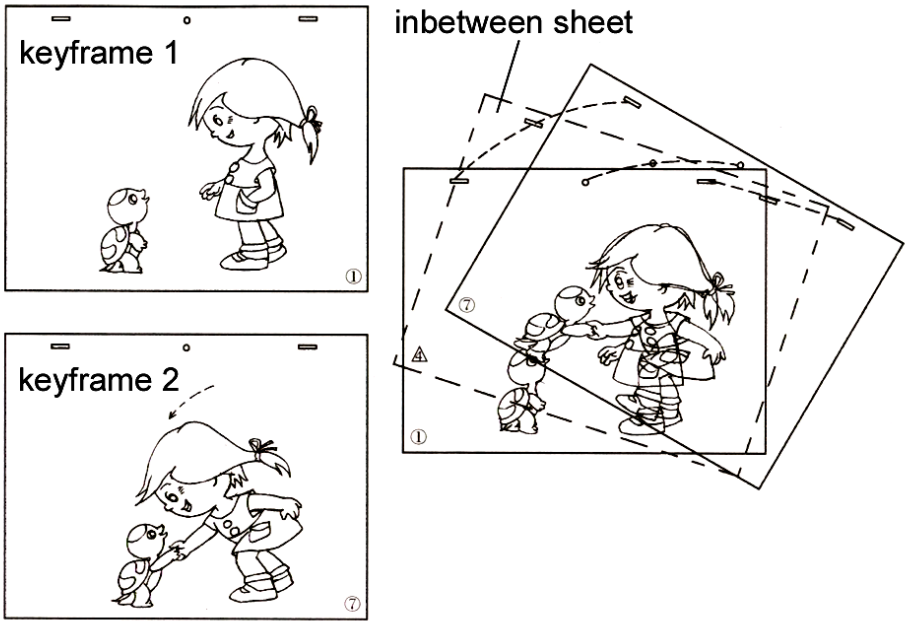


Figure 3.1: Conventional relocation method

is to estimate the *Displacement Vectors*, DVs, of the pixels in the two keyframes so that the correspondence between two frames is established for interpolation.

3.2.1 Feature-Based Matching Algorithm

3.2.1.1 Motion Estimation in Image Sequence

Using the pixel intensities as constraints for motion estimation, the given image sequence can be modelled as:

$$i(\mathbf{u}) = i'(F(\mathbf{u})) \qquad \text{(Eq. 3.1)}$$

where $i(\mathbf{u})$ is the intensity at point $\mathbf{u}$ in the current frame, i' is the intensity at a point in the next frame, $F(\mathbf{u})$ is a linear transformation function of image coordinates representing motion such as zooming, rotation and translation. When the motion is small between frames, zooming and rotation can be approximated by translation. Taking F as

a translation, Eq. 3.1 yields

$$i(\mathbf{u}) = i'(\mathbf{u} + \mathbf{d}) \qquad \text{(Eq. 3.2)}$$

Under this model, the intensity i' of the pixel at point $\mathbf{u}$+$\mathbf{d}$ in the next frame can be estimated from the intensity i of the pixel at point $\mathbf{u}$ in the current frame. Only the parameter $\mathbf{d}$, the *Displacement Vector* (DV), is needed and the problem reduces to motion estimation for all the pixels in the image.

Now the objective is to estimate the value of $\mathbf{d}$. The model can be linearized using the first order[1] Taylor series expansion of $i'(\mathbf{u} + \mathbf{d})$ about $\mathbf{d}$:

$$i(\mathbf{u}) = i'(\mathbf{u} + \mathbf{d}) = i'(\mathbf{u}) + \nabla i'(\mathbf{u})\mathbf{d}^T \qquad \text{(Eq. 3.3)}$$

where ∇ is the usual multi-dimensional gradient operator. Defining residual of the intensity r_i between two consecutive image at point $\mathbf{u}$, Eq. 3.3 yields:

(i) Residual of the intensity

$$r_i(\mathbf{u}, \mathbf{d}) = i(\mathbf{u}) - i'(\mathbf{u} + \mathbf{d}) \qquad \text{(Eq. 3.4)}$$

Apart from pixel intensity, the algorithm also incorporates other features for matching, i.e. the edgeness, r_e, the positive cornerness, r_p, the negative cornerness, r_n, the orientation smoothness, r_o and the displacement smoothness, r_d.

(ii) Residual of the edgeness

$$r_e(\mathbf{u}, \mathbf{d}) = e(\mathbf{u}) - e'(\mathbf{u} + \mathbf{d}) \qquad \text{(Eq. 3.5)}$$

Edgeness is defined as the magnitude of the gradient of intensity: $e = \left\| \frac{\partial i(\mathbf{u})}{\partial \mathbf{u}} \right\|$.

(iii) Residual of the positive cornerness

$$r_p(\mathbf{u}, \mathbf{d}) = p(\mathbf{u}) - p'(\mathbf{u} + \mathbf{d}) \qquad \text{(Eq. 3.6)}$$

[1] All higher order expansions are ignored.

(iv) Residual of the negative cornerness

$$r_n(\mathbf{u}, \mathbf{d}) = n(\mathbf{u}) - n'(\mathbf{u} + \mathbf{d}) \qquad \text{(Eq. 3.7)}$$

Corners are points where the gradient direction changes rapidly, and correspond to physical corners of objects. The cornerness of a point may be defined by instantaneous rate of change in the gradient direction along an edge curve that passes through the point. A sign is used to distinguish a corner of a white region in a black background (positive cornerness) from that of a black region in a white background (negative cornerness).

(v) Residual of the orientation smoothness

$$r_o(\mathbf{u}, \mathbf{d}) = \frac{\|\bar{\mathbf{d}} \times \mathbf{d}\|}{\|\bar{\mathbf{d}}\|} = \|\mathbf{d}\| \sin\theta \qquad \text{(Eq. 3.8)}$$

This is essentially the magnitude of the cross product between the displacement vector $\mathbf{d}$ and the average displacement vector $\bar{\mathbf{d}}$, where θ is the angle between the two vectors. It is a measure of the angle between the two vectors. When both vectors are in the same directions, their cross product is zero.

(vi) Residual of the displacement smoothness

$$r_d(\mathbf{u}, \mathbf{d}) = \left\|\bar{\mathbf{d}}(\mathbf{u}) - \mathbf{d}(\mathbf{u})\right\| \qquad \text{(Eq. 3.9)}$$

This is a measure of the magnitude of the difference vector between the estimated displacement vector and the average of its neighboring vectors.

The vector $\mathbf{d}$ is to be determined so that the weighted sum of squares of the residuals is minimized:

$$\sum_{\mathbf{u}} \{\lambda_i r_i^2(\mathbf{u}, \mathbf{d}) + \lambda_e r_e^2(\mathbf{u}, \mathbf{d}) + \lambda_p r_p^2(\mathbf{u}, \mathbf{d}) \qquad \text{(Eq. 3.10)}$$
$$+\lambda_n r_n^2(\mathbf{u}, \mathbf{d}) + \lambda_o r_o^2(\mathbf{u}, \mathbf{d}) + \lambda_d r_d^2(\mathbf{u}, \mathbf{d})\}$$
$$\rightarrow \min$$

where $\lambda_i, \lambda_e, \lambda_p, \lambda_n, \lambda_o$ and λ_d are weighting parameters that are dynamically adjusted at different resolution levels.

Defining

$$\mathbf{r} = [r_i, r_e, r_p, r_n, r_o, r_d]^T \tag{Eq. 3.11}$$

and

$$\Lambda = \text{diag}\{\lambda_i, \lambda_e, \lambda_p, \lambda_n, \lambda_o, \lambda_d\} \tag{Eq. 3.12}$$

which is the diagonal matrix with the corresponding diagonal elements, all the residue equations in the form of Eq. 3.4 can be noted as:

$$\Lambda \mathbf{r} = \Lambda \mathbf{J} \delta \mathbf{d} \tag{Eq. 3.13}$$

where $\mathbf{J}$ is the Jacobian matrix:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial i'}{\partial x} & \frac{\partial i'}{\partial y} \\ \frac{\partial e'}{\partial x} & \frac{\partial e'}{\partial y} \\ \frac{\partial p'}{\partial x} & \frac{\partial p'}{\partial y} \\ \frac{\partial n'}{\partial x} & \frac{\partial n'}{\partial y} \\ \frac{\bar{d}_y}{\|\bar{\mathbf{d}}\|} & -\frac{\bar{d}_x}{\|\bar{\mathbf{d}}\|} \\ \frac{\partial r_d}{\partial x} & \frac{\partial r_d}{\partial y} \end{bmatrix} \tag{Eq. 3.14}$$

and

$$\delta \mathbf{d} = \mathbf{d} - \mathbf{d}^* \tag{Eq. 3.15}$$

where $\mathbf{d}^*$ denote a nominal value of $\mathbf{d}$.

Starting with an initial estimation of a zero vector, $\mathbf{d}$, the DV can be iteratively updated by the following formula derived from Eq. 3.13:

$$\begin{gathered} \delta \mathbf{d} = \left(\mathbf{J}^T \Lambda^2 \mathbf{J}\right)^{-1} \mathbf{J}^T \Lambda^2 \mathbf{r} \\ \mathbf{d} \leftarrow \mathbf{d} + \delta \mathbf{d} \end{gathered} \tag{Eq. 3.16}$$

until no further reduction in error is observed.

3.2.1.2 Hierarchical Processing

A common problem for most search methods applied to image-to-image matching is the initial value problem. Since the evaluation function can be highly non-convex, the search method either needs good approximate values for the best mapping or needs to try out many different initial values in order to find a global optimum.

A hierarchical approach is applied to get the initial value of estimation. It derives series of increasingly smoothed images from the original images and uses these images or features detected in them for the matching. A pile of such increasingly smoothed and compressed images compose an *Image Pyramid*. The lowest level is the original image resolution. From level to level, the resolution and the size of the image is reduced and the image features are correspondingly brought closer together. Hence it will be relatively easy to find the optimal match.

The match of two smooth descriptions then supplies the initial values for the matching of two descriptions on the next level of resolution. This process is applied recursively to the series of image descriptions until the original unsmoothed descriptions can be matched using very good approximate values.

3.2.2 Modified Feature-Based Matching Algorithm

Seah [72, 71] makes several modifications and improvements to the original FBA. The modified algorithm, termed *Modified Feature-Based Matching Algorithm* (MFBA), improves upon the accuracy of matching by incorporating new mechanisms as well as enhancing the original algorithm. The various attributes, extracted from different types of features, invariably have different strength and ranges of values. All the different residual errors related to the attributes are normalized to the same scale by applying a set of weightages to them, cf. (Eq. 3.11). Respective weightages for the various constraints can be adjusted to reflect the relative importance of each attribute over the others. In

essence, the magnitude of the weights is not important, only the ratios of the weights among them are. From extensive experiments conducted for MFBA, a set of weightages is proposed that will cater for a wide range of input images to be processed, both line drawings and gray scale images. The general sets of weightages used for the gray scale images and line drawings are $\mathcal{L}_g = \{3\ 3\ 3\ 3\ 2\ 2\}$ and $\mathcal{L}_l = \{4\ 4\ 4\ 4\ 9\ 9\}$ respectively.

The overall matching algorithm can be summarized as follows:

(i) Extract the attribute images (intensity, edgeness, and cornerness) from the two input drawings at the original resolution (lowest level).

(ii) Recursively smooth these images for higher levels to generate lower resolution images.

(iii) Start with the highest level.

(iv) Initialize displacement vectors (DVs) at all pixel locations to zero.

(v) Iterate for a preset number of times, for all pixels.

  (a) Compute the average DVs and hence the orientation and displacement residuals.

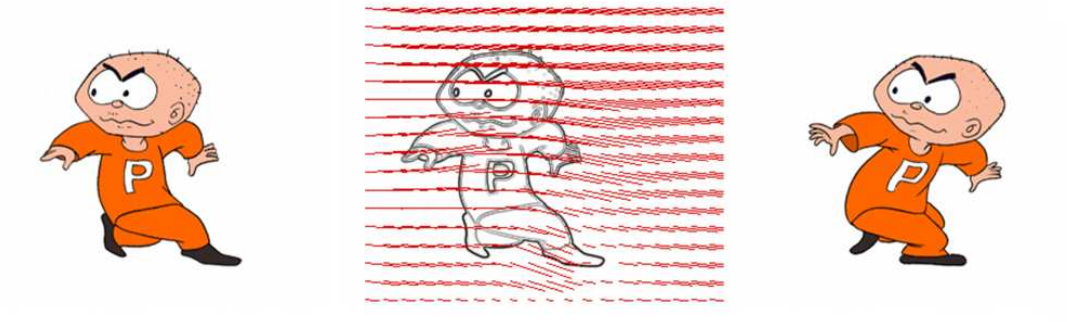
  (b) Compute and update the DVs.

(vi) Project the DVs downwards to the next lower level.

(vii) Set the level to the next lower level.

(viii) Repeat step (v) until the lowest level is reached.

All the DVs of pixels in the image compose the *Displacement Vector Field*, (DVF). Figure 3.2 illustrates two keyframes of a bear and the DVF obtained by matching them. The red DVs are superimposed on the source drawing, i.e. the first keyframe. Though every pixel of the drawing is associated with a DV, only a subset of the vectors are sampled at a regular interval for display.



Keyframe 1 DVF Keyframe 2

Figure 3.2: DVs between two keyframes computed by MFBA

3.2.3 Inbetween Generation by Interpolation

Based on DVF obtained from two keyframe drawings, linear inbetweening can be applied with complete automation. The general steps are as follows:

(i) Establish the DVF between the source and the target frames using MFBA.

(ii) Use linear interpolation to divide each DV from the DVF into the pre-set number of parts, that is, sub-DVs;

(iii) For each pixel in the source image, move it to its destination position in the inbetween frames, guided by the corresponding sub-DVs, d_{sub}, as follows:

$$P_i(m) = P_s(n) + \vec{d}_{sub}(n) \qquad m, n = 1, 2, ..., M \qquad \text{(Eq. 3.17)}$$

where $P_i(m)$ and $P_s(n)$ denote the m^{th} and n^{th} pixels in the inbetween image and source image, respectively. M is the total number of pixels of image, which remains constant in each pair of source and target images and their inbetweens. $\vec{d}_{sub}(n)$ denotes the sub-DV associated with the n^{th} pixel in the source image.

3.2.4 Handling Occlusion

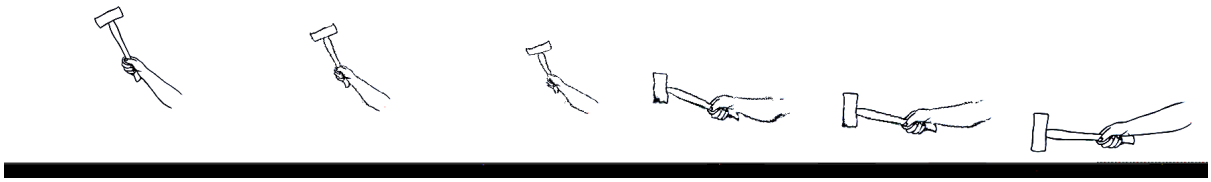
Occluded areas can be determined as proposed by Weng et. al. in [40]. Each point in the occluded areas is identified when no match for the point can be found in the corresponding image. The source occlusion map is generated by matching the destination image to the source image. Points where matches are found in the source image are marked as non-occluded locations. Those points not marked in this manner are therefore occluded. In the same way, the destination occlusion map can be constructed by matching the source image to the destination image. Hence, computation is increased by two fold when occlusion is to be resolved as an addition step has to be taken to match the destination image to the source image.

Once the occlusion maps are determined, in subsequent matching, points that fall in the occlusion area will not be considered. The DV therefore takes the value of the average DV at that location. In the same vein, occluded points will not contribute to the computation of average DV in a local vicinity. The occlusion maps are updated as the matching process continues.

3.2.5 Analysis

Compared with other existing methods that need an operator or designer to interact with or manually specify parameters into the system, MFBA achieves automatic inbetweening without any user-intervention. It handles a wide variety of cases, both line drawings and grayscale images, and it generates accurate result when the changes between keyframes are moderate.

Now let us examine an example with rather big change. In Figure 3.3, the two keyframes illustrate motion of a hand holding an ax. The intermediate four frames are inbetweens generated using MFBA. Obvious line distortion and jumping motion can be noticed in the inbetweens.



Keyframe 1 Inbetweens Keyframe 2

Figure 3.3: Inbetween generation using MFBA

The problem is due to the intrinsic limitation of MFBA. When establishing the image sequence model, it is assumed that the motion is not large so that zooming and rotation can be approximated by translation as shown in Eq. 3.2. The approximation of first-order Taylor expansion of $i'(\mathbf{u} + \mathbf{d})$ in Eq. 3.3 also suggests it is tenable with small motion because only when the displacements $\mathbf{d}$ is small can all higher order expansions be ignored. This is the common limitation of optical flow based methods. However, if all global motion is minimized, the condition of the formulas is met and MFBA will generate accurate matching result. Our proposed approach in the next section achieves the target.

3.3 Handling Global Motion by Keyframe Relocation

When the character involves big global changes e.g. translation and rotation, MFBA may not match accurately between two keyframes. To overcome the problem, we propose an approach to relocate keyframes in order to minimize global changes. In our approach, the poses of the character in the keyframes are first estimated. The keyframes are then

relocated by registering the poses. Inbetweens of the relocated keyframes are generated using MFBA and are back-transformed to the proper positions by interpolating the poses. It is assumed that a keyframe only contains a single character. This is reasonable since different characters are usually drawn in different layers in traditional animation for easy modification, as we described in Section 2.1.1.

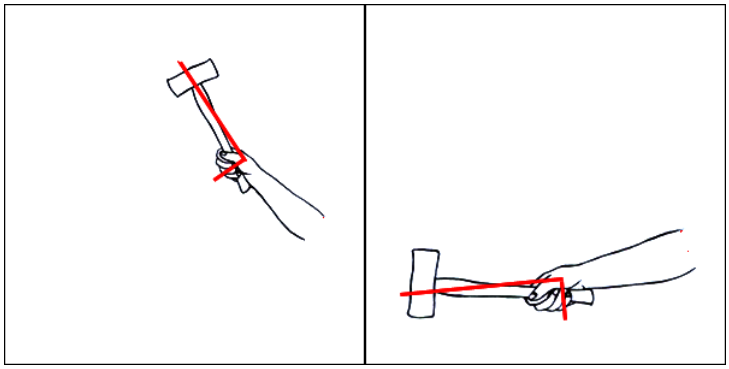
3.3.1 Pose Estimation

A character usually has many parts, each of which is approximately rigid, e.g. head, arms, legs, etc. We first apply our approach to such a part, referred to as a single object. Given two keyframes depicting the rigid movement of a single object, we first evaluate the approximate pose of the object using Principle Component Analysis, PCA [18].

Principle components can be used for describing boundaries and regions in a single image. The approach is to form two-dimensional vectors from the coordinates of the boundary of region. Vectors are formed from the coordinates of the pixels in an object if the region is to be described. It the boundary is to be described instead, only the coordinates of the points on the boundary are used. In our case, we use principle components to describe the boundary of a character only. Each pixel in the object is treated as a 2-D vector $\mathbf{x} = (a, b)^T$, where a and b are the x- and y-coordinate values of that pixel. These vectors are used to compute the mean vector and covariance matrix of the object.

As shown in Figure 3.4, for a rigid object, PCA gives a good evaluation of its pose i.e. the location, orientation and magnitude. The coordinates of the mean vector indicate the centroid or the location of the object. The two axes are the directions of the eigenvectors of $\mathbf{C}_x$ showing the object's orientation, and the eigenvalues are the variances along the axes denoting the object's size. Accordingly, the effect of translation is accounted for by centering the object about its mean. Rotation can be solved by aligning the object with its principal eigenvectors. The eigenvalues can be used for scaling. From these

parameters, the global motion of the object can be estimated. The magnitude obtained by PCA may be used to handle situations where a character is zoomed in or out in subsequent keyframes.



Keyframe 1 Keyframe 2

Figure 3.4: Pose estimation using PCA

3.3.2 Keyframe Relocation

The next step is to relocate the character by transforming the two keyframes to minimize the global changes. Obtaining the approximate pose of the object in the two keyframes as described in Section 3.3.1, the keyframes are transformed to make the object move to the same position with the same orientation and magnitude. We use the example with translation and rotation only. Scaling can be handled similarly. Suppose the pose parameters obtained are : $(center_1, \theta_1)$ for keyframe 1, and $(center_2, \theta_2)$ for keyframe 2, the following transformation is applied:

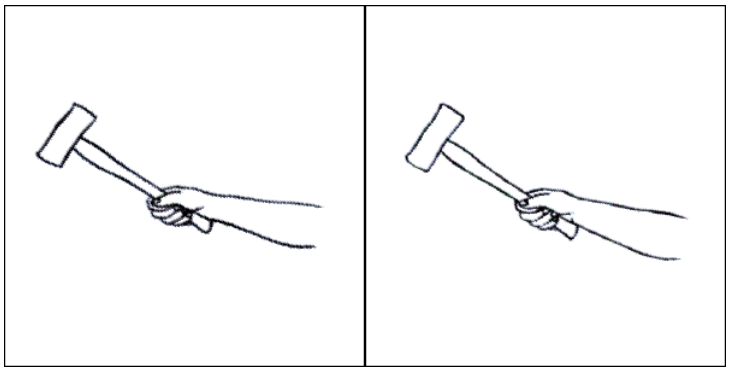
For keyframe 1:

$$Rotate(\frac{\theta_2 - \theta_1}{2}, center_1) \cdot Translate(\frac{center_2 - center_1}{2}) \qquad \text{(Eq. 3.18)}$$

For keyframe 2:

$$Rotate(\frac{\theta_1 - \theta_2}{2}, center_2) \cdot Translate(\frac{center_1 - center_2}{2}) \qquad \text{(Eq. 3.19)}$$

Rotate(θ, p) denotes rotation of angle θ round point p. *Translate*(l) denotes translating a point by l. Rotation is performed before translation. The result is shown in Figure 3.5.



Keyframe 1 Keyframe 2

Figure 3.5: Relocated keyframes in Figure 3.4

It can be seen in Figure 3.5 that by transformation, the arms in the two keyframes not only move to the same position in the canvas, but also point to the same direction. Applying PCA now will result in the same coordinates of the mean vector and the same directions of the eigenvectors. This indicates the global change is minimized. Hence the frames after transformation look almost the same. Local deformation is relatively small after transformation which satisfies the principle of MFBA thus the matching result is more accurate.

3.3.3 Pose Restoration

Now MFBA can work effectively on the local deformations. The result is shown in Figure 3.6 where four inbetweens are generated.

Next we need to restore the proper pose of the object in the inbetweens. Suppose n inbetweens are generated. For the i^{th} inbetween $(i = 1, 2, ...n)$, the following transforma-

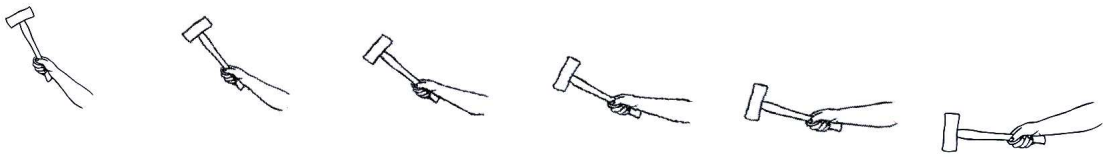
Keyframe 1 Inbetweens Keyframe 2

Figure 3.6: Inbetweens of keyframes in Figure 3.5 generated by MFBA

tion is applied:

$$Rotate((i - \frac{n+1}{2}) \cdot \frac{\theta_1 - \theta_2}{n+1}, \frac{center_1 + center_2}{2}) \cdot \quad \text{(Eq. 3.20)}$$
$$Translate((i - \frac{n+1}{2}) \cdot \frac{center_2 - center_1}{n+1})$$

The final result is shown in Figure 3.7. Figure 3.8 shows the overlapped sequence for better view of the motion.



Keyframe 1 Inbetweens Keyframe 2

Figure 3.7: Final inbetween result by pose restoration

Figure 3.9 shows another example comparing the results of applying MFBA only and applying global motion minimization. Figures 3.11, 3.13, and 3.12 show more results of generated inbetweens by global motion minimization. It can be seen that global motion is smoothly interpolated while local changes are also handled well, such as the hair and the dress of the girls and the wings of the bird. When relocation is applied, the keyframes are first transformed so that the global rotation is minimized thus the typical shortening problem in rotation is also minimized. Therefore, during each step of the method, the shape of the object is well retained.

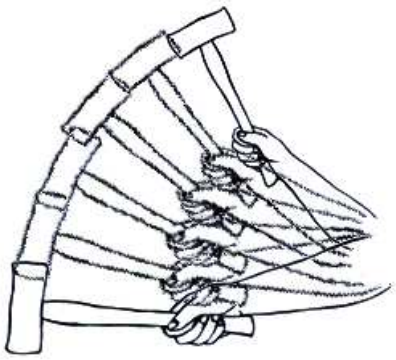


Figure 3.8: Superimposed frames in Figure 3.7

A major problem of pose estimation lies in determining the orientation using PCA. For instance, the face in the first keyframe in Figure 3.14 stretches in the second keyframe, where the results of PCA do not match. Advanced comparison scheme should be applied to evaluate the possible correspondence and choose the best matching, e.g. the characteristics of regions and the relationship between regions. This is another kind of high

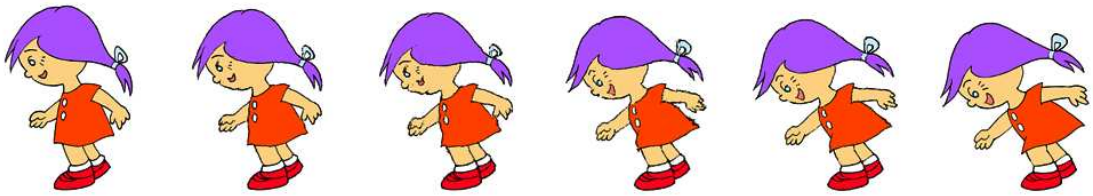
Keyframe 1 Inbetweens Keyframe 2

(a) MFBA

Keyframe 1 Inbetweens Keyframe 2

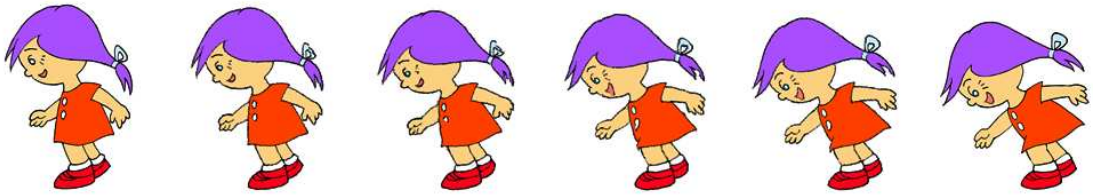
(b) MFBA + global motion minimization

Figure 3.9: Inbetween generation results: fish



Keyframe 1 Inbetweens Keyframe 2

(a) MFBA



Keyframe 1 Inbetweens Keyframe 2

(b) MFBA + global motion minimization

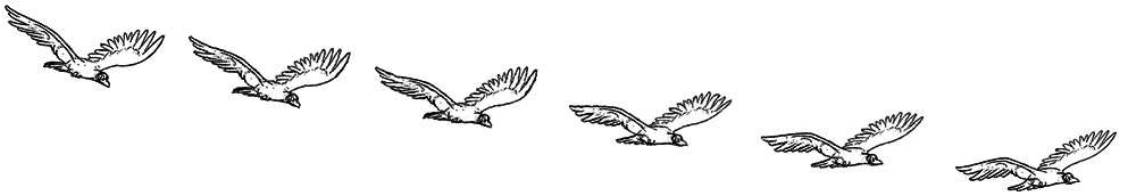
Figure 3.10: Inbetween generation results: girl

Inbetween 2 Inbetween 3 Inbetween 2 Inbetween 3

(a) MFBA (b) MFBA + global motion minimization

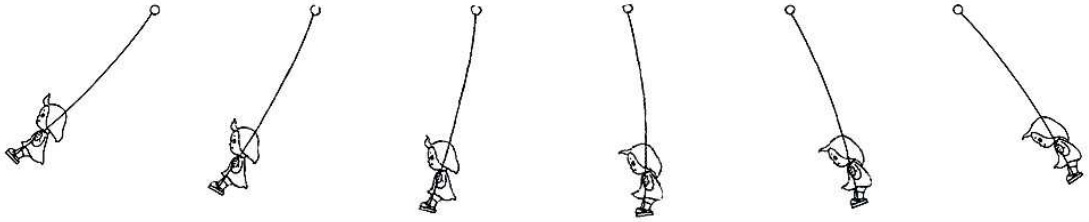
Figure 3.11: Detail comparison of Figure 3.10
(Main distortion circled)

level information which may be further researched in the future.



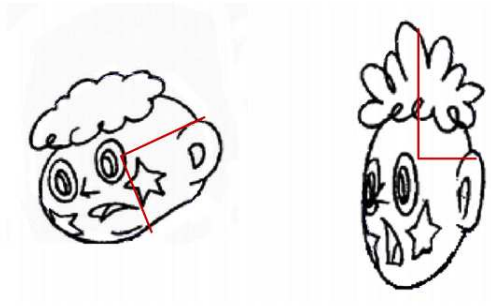
Keyframe 1 Inbetweens Keyframe 2

Figure 3.12: Inbetween generation results: bird



Keyframe 1 Inbetweens Keyframe 2

Figure 3.13: Inbetween generation results: swinging



Keyframe 1 Keyframe 2

Figure 3.14: Problem of determining orientation

3.4 Inbetween Generation Based on Character Segmentation

Our current algorithm takes the overall location and orientation of the character into account and thus minimizes its global changes. When various parts of the character involve

different drastic changes, it cannot be directly handled by global motion minimization. For instance, when a person stoops down, his upper body rotates sharply while the lower part remains almost still. In this case, no matter how the keyframes are relocated, the changes remain large. However, the components e.g. head, arms, torso and legs, remain approximately rigid during the movement. In traditional animation, when this happens, the common practice is to draw the different parts respectively, each using the *relocation* approach.

Similarly, the aim of our computer-assisted approach is to break down the character into components which are consistent with the structure of the character thus can be regarded as approximately rigid. It is difficult to achieve this using existing image segmentation methods based on image features e.g. gradient or edge detection, since no structural information of the character is taken into account. Here we propose a novel technique for character segmentation based on skeleton of the character. After appropriate segmentation, same components in the keyframes are corresponded using a matching scheme. Then each component can be treated as a single object and handled separately using the approach we proposed in Section 3.2 and 3.3.

3.4.1 Skeleton Extraction

To appropriately segment a character into several approximately rigid components, we apply a method of skeleton representation. A skeleton is used for image representation in a compressed form while abstracting the complicated underlying transformations. It captures the overall frame of the complex character by using a simple set of parameters and eliminates the details. In 3D animation, skeleton technique is commonly used for such purpose. For 2D animation, in some previous work [58, 48, 54] a user-defined skeleton is manipulated to drive inbetween sequences. In our approach, skeleton representation is used to assist appropriate segmentation.

3.4.1.1 Gap Closing

We will extract the skeleton according to the boundary of a character. In keyframes of line drawings, the contour of a character usually forms a closed region. It is critical and compulsory for the next production procedures such as coloring. Coloring can only be applied successfully for closed regions. If a region is not closed, the applied color will 'leak' out of the region. However, some small gaps of broken lines can be introduced either from the scanning process or from original line drawings due to the nature of the artist's style of drawing. To resolve the problem, Seah's gap-closing algorithm [70] is applied to bridge broken lines.

3.4.1.2 Boundary Tracing

To cover a wide range of input images, including line drawings as well as color images, the classic border tracing algorithm [52] is adapted to our application. The main steps are as follows:

(i) Search the image from left to right and from top to bottom until a non-background pixel is found; this pixel P_0 then has the minimum row value of all the contour pixels. Pixel P_0 is the starting pixel of the contour. Define a variable *dir* which stores the direction of the previous move along the contour from the previous contour element to the current one. For 8-connectivity, *dir* is set as shown in Figure 3.15. The initial *dir* is pre-set to zero.

(ii) Search the 3×3 neighborhood of the current pixel in a clockwise direction, beginning the neighborhood search in direction:

$$dir = (dir + 3) \mod 8 \qquad \text{(Eq. 3.21)}$$

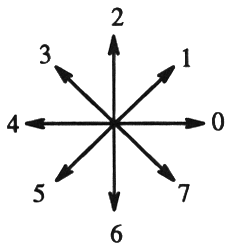


Figure 3.15: Direction definition of 8-connectivity

The first pixel found with non-background value is a new contour element P_n. Update the *dir* value.

(iii) If the current contour element P_n is equal to the first contour element P_0, stop. Otherwise repeat step (ii).

Figure 3.16 illustrates the route of the contour tracing.

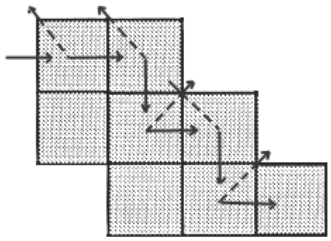


Figure 3.16: Route of contour tracing
(The solid lines show the tracing route; the dashed ones show the starting *dir* of the current step based on the previous one.)

Since we assume one character in one frame, this algorithm traces the character contour over and represents it by a sequence of pixels $P_0...P_{n-1}$.

The algorithm is applicable on the following conditions:

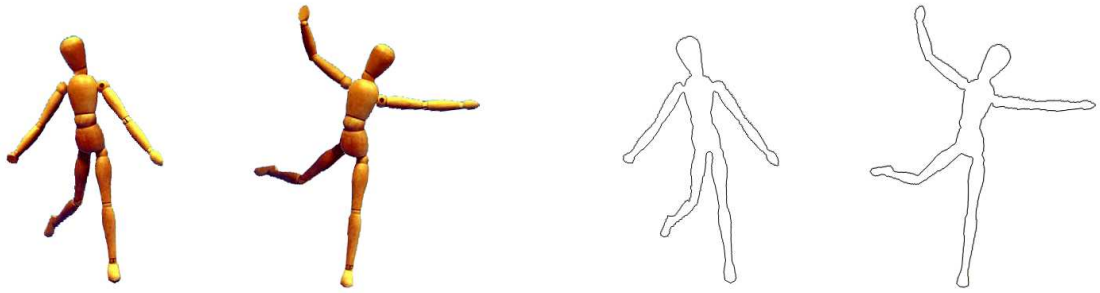
(i) The boundary of the character must be closed.

(ii) The first pixel found must belong to the boundary rather than some noise stroke.

The conditions can be met by the pre-processing of the input images described in the previous section.

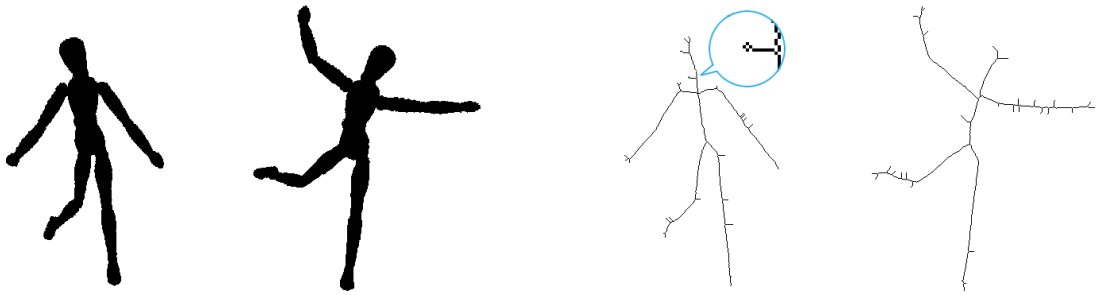
3.4.1.3 SUSAN Thinning

The skeleton of a character can now be obtained by thinning. The interior details inside the boundary are removed by the scan line seed fill algorithm [66]. SUSAN thinning algorithm [51] is applied. SUSAN is an acronym for Smallest Univalue Segment Assimilating Nucleus. It is originally proposed for binary thinning, which is applied to edge images. The thinning rules of the algorithm are summarized in [51], which are simple and fast to carry out, requiring only one pass through the image. It ensures one-pixel thickness and connectivity of the resulting skeleton. Figure 3.17 shows the process step by step.



(a) Original keyframes

(b) Contour tracing



(c) Interior removal

(d) Thinning

Figure 3.17: Skeleton extraction

3.4.1.4 Skeleton Smoothing

It can be noticed that the skeleton in Figure 3.17(d) contains a large number of artefacts. Since we assume the major components e.g. the head, the arms, the torso and the legs, to be roughly rigid, the artefacts are redundant. They will only lead to trivial parts when segmenting the character and greatly increase the complexity of matching between two keyframes. There are also some undesired holes in the skeleton, as magnified in Figure 3.17(c), which cannot be removed by a general pruning process.

To solve the problem, we take the following steps. After the boundary of the original image is traced, the boundary image is first reduced in size thus smoothed. In this way, the result will have fewer artefacts after SUSAN thinning is applied. The thinning also performs faster now that the size of image is reduced. An iterative pruning process follows to further trim off the artefacts. The skeleton in reduced size is then enlarged to the original image size. Additional thinning is applied to guarantee one pixel thickness, which is required for branch tracing in the next step. Iterative pruning is also applied once more to the enlarged skeleton until the length of all branches exceeds a pre-set threshold based on analysis of broad samples of real drawings in animation production. Figure 3.18 shows the final skeleton. Notice the numbers of branches are not necessarily the same for two keyframes. There is slight difference between this skeleton and the one obtained directly using SUSAN thinning to the original image, but it is adequate to capture the approximate structure of the character for our use.

3.4.2 Character Segmentation

3.4.2.1 Branch division

The skeleton obtained from thinning is a raster image and need to be traced and divided into branches of sequential points. The Depth-first search algorithm is adopted. Defining:

- junction point - a point with more than two neighboring points; and

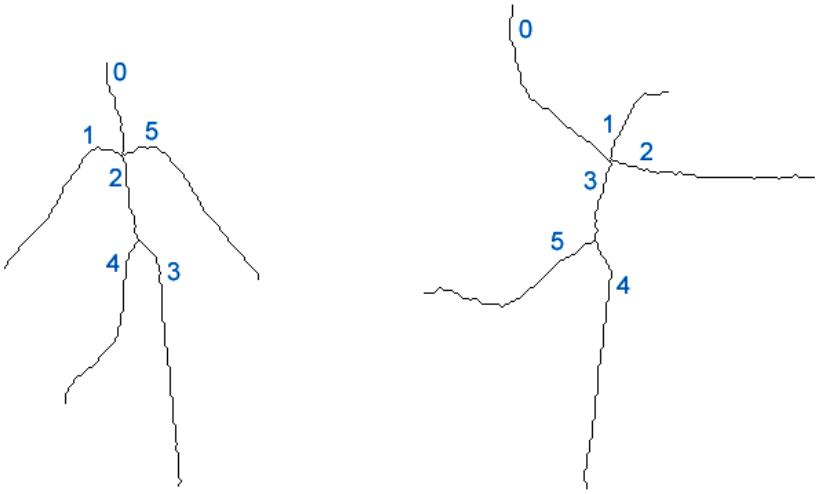


Figure 3.18: Smooth skeleton

- end point - a point with only one neighbor;

the search starts from any one of the end points and traces down the neighboring points. Once a junction point is encountered, the searched points will be grouped as a branch. Finally the whole raster skeleton is searched and divided into n branches: $\{B_0, B_1, ..., B_{n-1}\}$. Each branch B_i is composed of m_i pixels in order: $B_i = \{p_0^i, p_1^i, ..., p_{m_i-1}^i\}$. Figure 3.18 shows the result with the branch indices.

3.4.2.2 Component Segmentation

The next step is to decompose the character into components based on the skeleton. For each non-background pixel P_i in the image, its distance $D_{i,j}$ to the branch B_j is calculated. Suppose there are m branches in the skeleton. We calculate the distance from P_i to each branch: $D_{i,0}$, $D_{i,1}$, ..., $D_{i,m}$. P_i will be associated to B_k to which the distance $D_{i,k}$ is the smallest. In this way, each non-background pixel will be associated with a branch of the skeleton and the original image is segmented into m components accordingly.

The result of segmentation is illustrated in Figure 3.19. The character is split for clearer view.

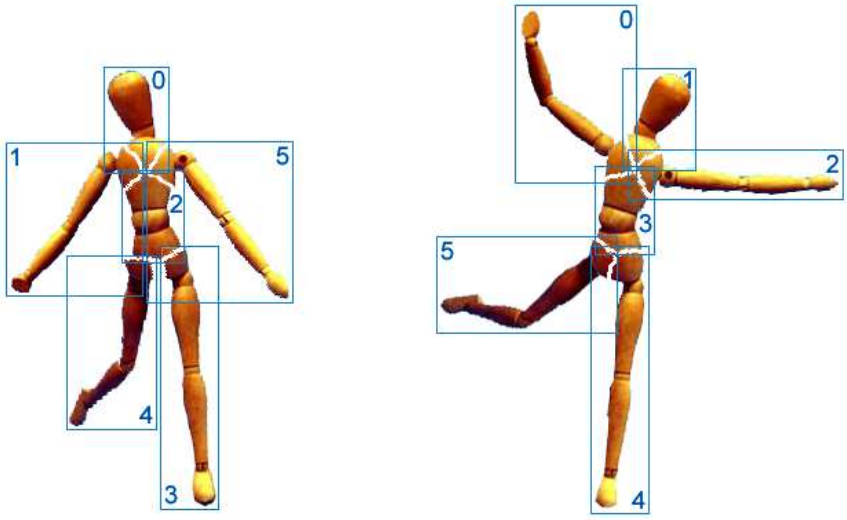


Figure 3.19: Skeleton-based segmentation

3.4.3 Skeleton Correspondence

To correspond the branches indicating the same components in the keyframes, we propose a graph matching algorithm, which is similar to some existing methods [75, 56] but directly matches the branches instead of nodes.

3.4.3.1 Attributes of A Graph

Neighboring relationship of the branches is selected as a feature for matching because in most cases of a character's movement, the topology of the components i.e. the relations among head, neck, torso and two arms of the character, remains relatively stable. For each end of a branch B_i, the index of every neighboring branch counterclockwise is recorded. We utilize this feature to select the matching candidates at every step.

Another feature of matching is the length of the branches. Since each component is reduced to a branch of the skeleton, the length of the branch indicating the same

component should remain roughly the same. The length difference of two corresponded branches will be calculated to evaluate a match. The aim is to find a match in which the total difference in branch length is minimized.

If the numbers of branches in two keyframes are not the same, the one with fewer branches is always regarded as the source graph G_{src} and the other as the target G_{tgt}.

3.4.3.2 Graph Matching

Generally, longer branches represent main components and are more steady between keyframes than shorter ones. Accordingly our search for a match starts with the longest branch B_j^{src} in G_{src}. The candidates in G_{tgt} are put into a stack, among which the longest branch B_k^{tgt} will be taken as the current match to B_j^{src} and the difference of their length is computed as d_1. Then the ordered neighboring branches of both B_j^{src} and B_k^{tgt} are put into stacks as candidates. The first branch in each stack are taken out and matched. The difference of their length d_2 is cumulated. The process continues until all the branches in G_{src} find a correspondence in G_{tgt} or there is no candidate left in G_{tgt}. This constitutes a single pass in which a total length difference of the matched branches is obtained: $D_{pass_0} = \sum_{i=0}^{p} d_i$. During the process, there may be more than one candidate in G_{tgt} for current branch in G_{src}, thus the steps can be traced back to result in new passes. In case the topology of branches is not the same in the keyframes, candidates which are not among the neighbors of currently matched branches are also allowed. A best match is found when D_{pass_i} is minimized. For the skeleton in Figure 3.18, the best matching result is

$0 \Rightarrow 1, 1 \Rightarrow 0, 2 \Rightarrow 3, 3 \Rightarrow 4, 4 \Rightarrow 5, 5 \Rightarrow 2.$

The size of the original images is 350×400 pixels. The smallest difference of branch length is 971 pixels. The direction of matching is also recorded indicating which ends of the branches match to another. Since each branch is related to a component, the

components in two keyframes are thus matched accordingly.

The overall steps of character segmentation and matching are as follows:

- Extract the skeletons from the original frames;
- Divide the skeletons into branches;
- Segment the character into components according to the branches;
- Match the branches in two keyframes.

3.4.4 Interpolation Based on Segmentation

To estimate and minimize the global movement of each component, a branch B_i is approximated by a straight line linking the two ends p_0^i and p_{n-1}^i. A global transformation from the source component to the target one can thus be calculated and the components in the two keyframes are relocated to minimize the global movement, as illustrated in Figure 3.20(a). Since the motion between the keyframes becomes smaller which satisfies the evolution of the equations in Section 3.2.1, MFBA will give more accurate motion estimation and generate smoother inbetweens as in Figure 3.20(b). After the inbetweens are generated, they are back-transformed in an interpolative way in order to be re-positioned with the right pose, as shown in Figure 3.20(c).

The inbetween result of the keyframes in Figure 3.17(a) is shown in Figure 3.21(b), compared with the previous result using MFBA only in Figure 3.21(a). In each sequence, the first and the last are the two keyframes. The intermediate four frames are inbetweens generated. It is obvious that the motion of the character's various components is interpolated more smoothly.

Another example of line drawing has been tested as shown in Figure 3.22 and 3.23. When calculating the shortest distance from pixels to branches, there is no pixel whose

(a) Components in original keyframes (b) Relocated components

Keyframe 1 Inbetweens Keyframe 2

(c) Inbetweens of relocated keyframes

Keyframe 1 Inbetweens Keyframe 2

(d) Restored inbetweens

Figure 3.20: Global motion minimization

distance to branch 1 is the shortest in keyframe 1. Therefore segment 1 in keyframe 1 is empty and not depicted in Figure 3.22(c). The best matching result is

$0 \Rightarrow 0, 1 \Rightarrow 2, 2 \Rightarrow 1, 3 \Rightarrow 3, 4 \Rightarrow 5, 5 \Rightarrow 4, 6 \Rightarrow 6.$

Two more examples are illustrated in Figure 3.24 and Figure 3.25. From all the results, it is obvious that the motion of the inbetweens is smoother and the shapes of various components are well retained.

Keyframe 1 Inbetweens Keyframe 2

(b) Global motion minimization as single object

Keyframe 1 Inbetweens Keyframe 2

(b) Segmentation-based global motion minimization

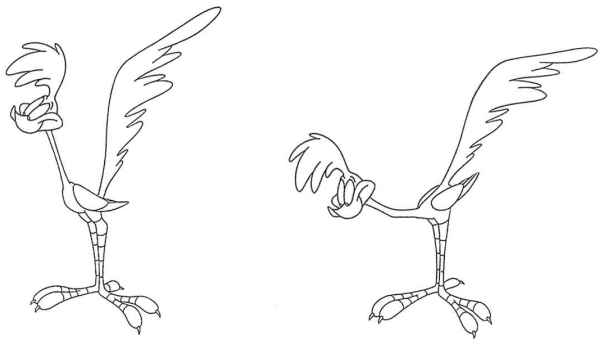
Figure 3.21: Inbetween generation results: puppet

3.4.5 Gap Filling

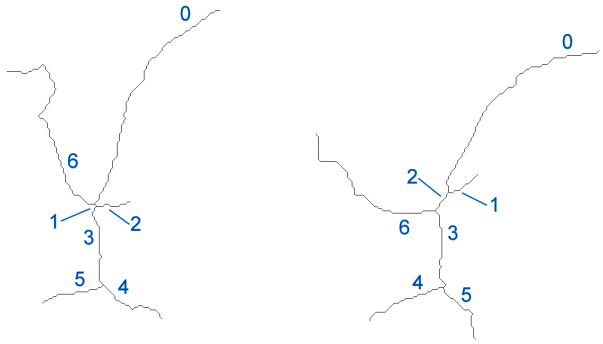
The problem of Figure 3.21 is some gaps around the joints in the inbetweens, which can be noticed more clearly as zoomed in in Figure 3.26.

Suppose we have two keyframes K_1 and K_2 and the DVs computed using our methods proposed in previous sections. Now the DVs will be used to generate inbetweens, e.g. I_m. Two adjacent pixels P_i and P_j which are segmented into different components in K_1 may not be neighbors in I_m as shown in Figure 3.27. Consequently a gap occurs between P_i and P_j in I_m. Line drawings are only comprised of strokes without regions of greyscale or color thus there are no distinct gaps. The problem is more noticeable in greyscale or colored images such as Figure 3.26.

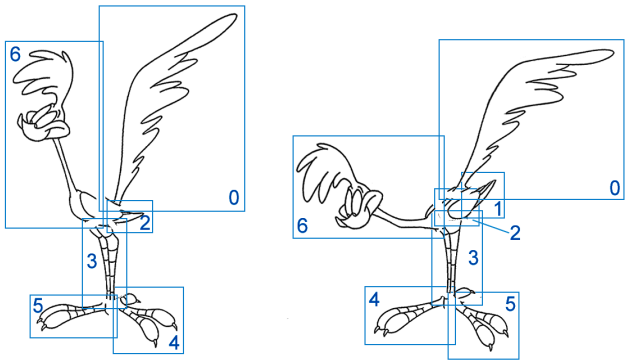
In current method, inbetweens of each component is generated independently without considering its relationship to others. No constraint is forced upon adjacent segments to



(a) Original keyframes

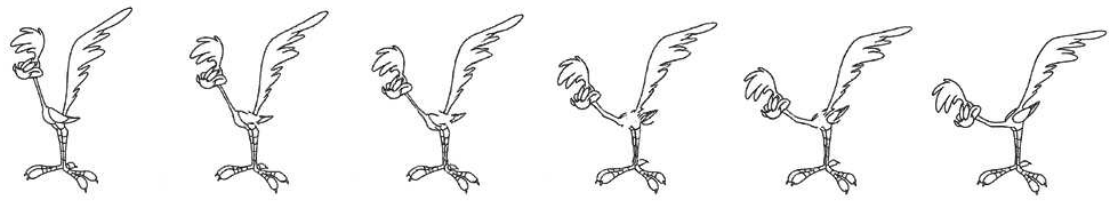


(b) Smooth skeleton



(c) Skeleton-based segmentation

Figure 3.22: Skeleton extraction and character segmentation: roadrunner



Keyframe 1 Inbetweens Keyframe 2

(b) MFBA + global motion minimization

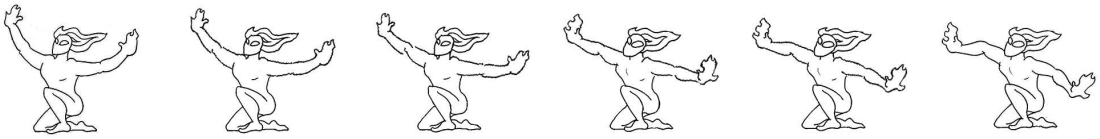
Figure 3.23: Inbetween generation results: roadrunner

Keyframe 1 Inbetweens Keyframe 2

(b) MFBA + global motion minimization

Figure 3.24: Inbetween generation results: boy

maintain continuity and ensure smoothness in intermediate frames automatically generated. Thus the discontinuity of the movement of neighboring pixels around the joints produces gaps. Each gap is a consequence of a jump of the values of two neighboring DVs. Thus, gaps can be detected by detecting the jumps of adjacent DVs in the DVF.



Keyframe 1 Inbetweens Keyframe 2

(b) MFBA + global motion minimization

Figure 3.25: Inbetween generation results: fireman

Figure 3.26: Gaps between joints of segments

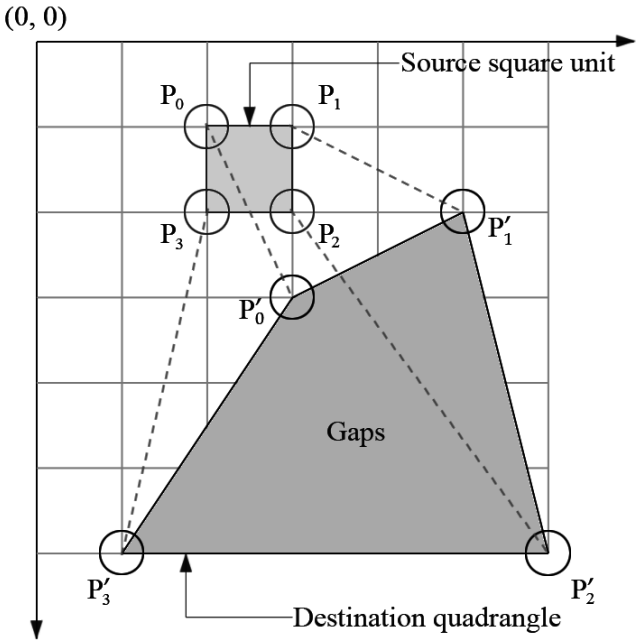


Figure 3.27: DV jump detection

To handle the problem, we apply the DVGI, DV-guided interpolation, proposed in [36]. The algorithm is based on DVs between two raster keyframes, which are used to generate intermediate frames. With the guidance of DVs, the algorithm classifies the gaps into several categories and handles them accordingly. A 2×2 square window is moved on the source image, e.g. K_1. The square is made up of 4 adjacent pixels and after moving to their new positions in the target image e.g. I_m, they form a new quadrangle. It may

be stretched or squeezed to yield irregular quadrangles. A stretched quadrangle contains gaps and is taken as the unit of gap filling. Every square unit in K_1 is scanned and each resulting stretched quadrangle in I_m will be filled by a brute force gap filling method that eliminates every kind of gap in it. For a pixel on the edges of the quadrangle or inside it, it color is calculated by the weighted averaging of the four vertices of the quadrangle:

$$\begin{cases} \overline{R}_i = \frac{\sum_{j=0}^{3} \frac{R_j}{d_{i,j}}}{\sum_{j=0}^{3} \frac{1}{d_{i,j}}} \\ \overline{G}_i = \frac{\sum_{j=0}^{3} \frac{G_j}{d_{i,j}}}{\sum_{j=0}^{3} \frac{1}{d_{i,j}}} \\ \overline{B}_i = \frac{\sum_{j=0}^{3} \frac{B_j}{d_{i,j}}}{\sum_{j=0}^{3} \frac{1}{d_{i,j}}} \end{cases} \quad \text{(Eq. 3.22)}$$

where $\overline{R}_i$, $\overline{G}_i$ and $\overline{B}_i$ are the color channel values used to fill a pixel. R_i, G_i and B_i are the values of the vertices of the quadrangle. $d_{i,j}$ is the distance between the j^{th} vertex and the i^{th} pixel to be filled.

Quadrangles could also be overlapped in I_m, i.e. gap pixels belonging to the quadrangle currently being scanned might have already been filled by a previously processed quadrangle. To handle the issue, the reliability of filling pixels is classified into 7 levels. It ensures a newly computed color value of a gap pixel with a higher reliability level overrides the older one with a lower level. This mechanism is developed to increase the accuracy and reliability of gap filling. The algorithm ensures 100% gap-filling coverage. In the end, a gap-free image is guaranteed.

The final inbetween result is shown in Figure 3.28 and Figure 3.29.

3.5 Summary

In this chapter, we proposed several techniques to solve the problem of inaccurate keyframe matching in the area of image-based inbetween generation. We first proposed a novel technique to handles global and local motion of a character respectively. To

Keyframe 1 Inbetweens Keyframe 2

Figure 3.28: Frames in Figure 3.21 after gap filling

Figure 3.29: Parts in Figure 3.26 after gap filling

tackle global motion, approximate pose of the character in each keyframe is estimated and global motion is computed accordingly. The keyframes are then relocated to minimize the global motion. To handle local motion, MFBA is applied, which estimates the motion of pixels using various image features. Our approach achieves automation and can cope with a wide range of cases in image matching, both for line drawings and gray scale images. For a complicated character, we proposed a technique to segment it into components based on its automatically extracted skeleton. Then graph matching is applied to match the skeleton branches as well as the components in successive keyframes. Our first technique proposed in this chapter is subsequently applied to each component and handles its global and local motion respectively. A gap filling scheme is applied to remove gaps around the joints of components. The results show promising improvement in accuracy of keyframe matching and smoothness of the generated inbetweens.

Chapter 4

Inbetween Generation for Vector Representation

4.1 Introduction

The techniques in animation production develop rapidly, in which vector-based animation is becoming increasingly popular in industry. It has several advantages over traditional animation drawn on paper. Vector-based drawing facilitates easy modification and correction. It supports free scaling without quality loss, which makes the production independent of output resolution. Thus it is suitable for high definition animation, which is a trend for animation industry. In addition, vector form reduces storage greatly, compared with huge piles of drawings on paper or larger size of raster images. Therefore, it is easy and efficient for transfer, which is very important to current industry where cooperations cross over the world.

There are mainly two approaches of stroke representation among existing vector-based animation systems. One is to use curves to represent strokes of constant width. This limits the drawing of animators and the drawings may look non-artistic, mechanical or even dull. The other is to use regions representation, which is general and can represent a wider ranges of strokes, including those with variable width within the same stroke. However, it is difficult to handle inbetween generation using the representation, because

interpolating two regions requires significant amount of knowledge on how various parts of the two regions are corresponded.

A novel representation of strokes, disk B-spline curve ($DBSC$), has been proposed in [92, 90, 73]. It is efficient in storage, easy to manipulate and capable of mimicking various styles of free drawing. When user draws on a digital tablet, a stroke is represented by a DBSC in real-time. To create animations, two keyframes, each containing a set of DBSCs, are given and inbetweens are automatically generated by interpolating corresponding DBSCs. However, in previous method [90, 73], linear interpolation is applied to points evenly taken in parametric domain of two DBSCs. No properties of the drawing e.g. shapes or motion of the character, are considered, which may result in distortion and unrealistic motion in animation.

In this chapter we will investigate various effective information in the keyframe drawings, extract it and employ it in inbetween generation, in order to handle the aforementioned problems.

4.2 Disk B-Spline Curves

4.2.1 Definition

A disk B-spline curve defines a region as well as its center curve. It is a generalization of disk Bézier curve and can be used as an effective representation of freeform 2D shapes. With rigid mathematical fundamentals, it is flexible for manipulation and deformation and it uses small dataset. More details can be found in [92, 90, 73]. Similar approach is proposed in [74] using an interpolatory interval spline curve defined by control shape to approximate artistic brushstrokes.

Defining $N_{i,p}(t)$ as the i^{th} B-spline basis [44] of degree p with knot vector:

$$[u_0, ..., u_m] = \{\underbrace{a, ...a}_{p+1}, u_{p+1}, ..., u_{m-p-1}, \underbrace{b..., b}_{p+1}\} \qquad \text{(Eq. 4.1)}$$

and a disk [45] in the plane:

$$< P; c >= \{x \in R^2 | \mid x - c \mid \leq r, c \in R^2, r \in R^+\} \quad \text{(Eq. 4.2)}$$

a disk B-spline curve is defined as follows:

$$< D > (t) = \sum_{i=0}^{n} N_{i,p}(t) < P_i; r_i > \quad \text{(Eq. 4.3)}$$

where P_i is the control point, r_i is the control radius, and $m = n + p + 1$.

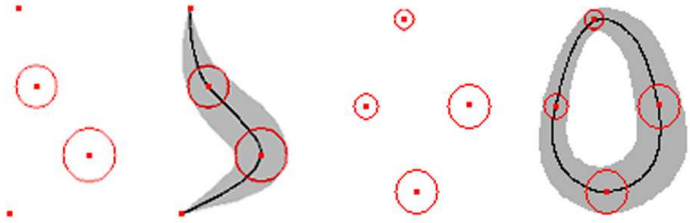
A DBSC can be viewed as two parts: a center curve (a B-spline curve) and a radius function (a B-spline scalar function). Therefore most of its properties and algorithms can be obtained by applying B-spline curve functions to the two parts respectively.

When a group of points $\{Q_i\}, (i = 0, ..., m)$ on the central curve and their corresponding maximum distances $\{c_i\}$ are given, a DBSC can be obtained by interpolation. Its center curve passes these data points $\{Q_i\}$ and maximum radii are $\{c_i\}$. The knot vector of a disk B-spline curve is determined using the chord length method. B-spline curve interpolation and scalar function method are used to obtain DBSC interpolation as shown in Figure 4.1. The curve can be either closed or open.

Given the data points and their radii in Figures 4.1(a) and (c), a smooth shape can be represented using either an open or closed DBSC. For degree-three DBSCs which are used in subsequent sections in this thesis, at least four data points are required for interpolating a curve. With the flexible stroke representation of DBSC, user can directly draw on a tablet and the strokes are in real-time represented by DBSCs.

4.2.2 Linear Interpolation

To illustrate the interpolation of two DBSCs more clearly, we assume the two DBSCs have the same degree, number of control points and knot vector i.e.



(a) (b) (c) (d)

(a) Data points (The radii of two end points are 0)

(b) An open DBSC by interpolating the data points in (a)

(c) Data points (The top point is both the starting and the ending point)

(d) A closed DBSC by interpolating the data points in (c)

Figure 4.1: Disk B-spline curve

$$< D_1 > (t) = \sum_{i=0}^{n} N_{i,p}(t) < P_i; r_i > \qquad \text{(Eq. 4.4)}$$
$$< D_2 > (t) = \sum_{i=0}^{n} N_{i,p}(t) < Q_i; s_i >$$

The in-between DBSCs can be generated by interpolating $< D_1 >$ and $< D_2 >$'s corresponding control points and radii. Given the control points P_j, Q_j and the control radii r_j, s_j, to get m in-between DBSCs by linear interpolation, the result is

$$\begin{cases} P_j^i = (1 - \frac{i+1}{m+1}) \cdot P_j + \frac{i+1}{m+1} \cdot Q_j \\ r_j^i = (1 - \frac{i+1}{m+1}) \cdot r_j + \frac{i+1}{m+1} \cdot s_j \end{cases} \qquad \text{(Eq. 4.5)}$$

where $i = 0, 1, ..., m-1$.

When two disk B-spline curves have different degrees, the curve with lower degree is converted into a higher degree curve through elevation algorithm.

For two curves with different numbers of control points and knot vectors, different approaches may be adopted for different situations. When user draws strokes on a tablet, the density of recorded data points can be changed by adjusting the tablet properties. If the density is high, the following approach is applied. The curve with more control points, S_1, remains unchanged. Supposing S_1 has n control points and its knot vector is $T_1 = \lfloor t_0, ..., t_{n+p} \rfloor = \{a, ...a, t_{p+1}, ..., t_{n-1}, b, ..., b\}$, we compute points G_i and radii e_i on the other curve S_2 at t_i $(i = p, p+1, ..., n)$. By interpolating G_i and e_i, a new curve, S_2', of n control points and knot vector T_1 is obtained, which will be used to generate inbetweens with S_1. Since the sampling density is high, it is sufficient that S_2' is very close to S_2 and smooth intermediate curves can be generated. If the sampling density of the tablet is low, the union of the two knot vectors of S_1 and S_2, T_{12}, is computed. Points and radii on T_{12} are computed for both S_1 and S_2. Two new curves, S_1' and S_2', are obtained by interpolating the computed points and radii and used to generate intermediate curves. This approach guarantees that S_1' and S_2' are close to S_1 and S_2 respectively and smooth interpolation can be achieved.

When two curves have the same numbers of control points but different knot vectors, the same method of using the union of the two knot vectors as described above is used.

When creating animations, given two keyframe drawings composed of a number of DBSCs, the inbetweens are generated by interpolating the corresponding pairs of DBSCs. The correspondence is established by the order of drawing on the tablet or specified by users. An example is shown in Figure 4.2, which is automatically generated using a DBSC-based drawing system [90].

4.2.3 Nonlinear Interpolation

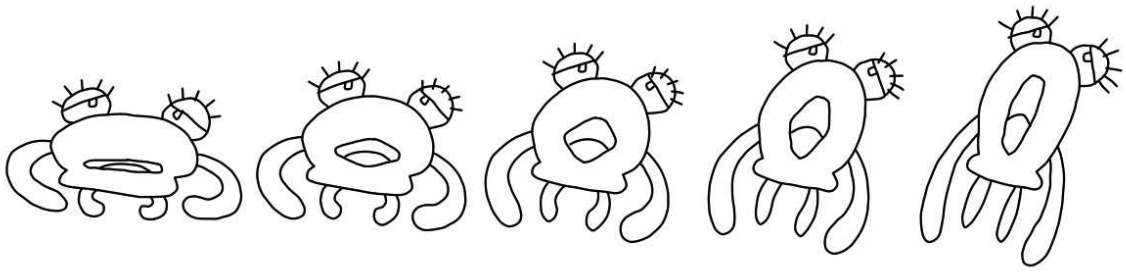
According to one of the essential animation principles 'slow in and slow out', linear interpolation is far from adequate to simulate real life motions. It usually results in

Keyframe 1 Inbetweens Keyframe 2

Figure 4.2: Inbetween generation using DBSC

dull and lifeless motion. In Figure 4.3, the motion is generated by linear interpolation. Suppose n inbetweens are to be generated. When interpolating point P_i in keyframe 1 and Q_i in keyframe 2, the corresponding intermediate point T_i is computed based on even intervals as follows:

$$T_i^k = (1 - \frac{k}{n+1}) \cdot P_i + \frac{k}{n+1} \cdot Q_i \qquad \text{(Eq. 4.6)}$$



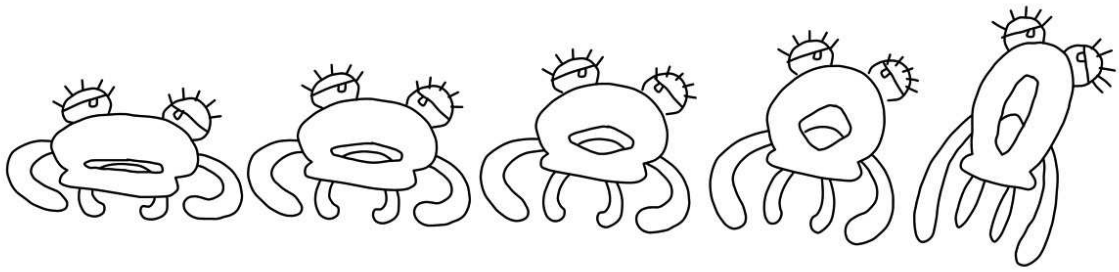
Keyframe 1 Inbetweens Keyframe 2

Figure 4.3: Linear interpolation: jellyfish

When the resulting sequence is played, the character moves with a constant speed, which looks very mechanical. In animation, a character usually moves with a variant speed, which vividly mimics movement in real life. To achieve uneven changing of shape and motion, the linear interpolation can be modified as follows:

$$T_i^k = (1 - m) \cdot P_i + m \cdot Q_i \qquad \text{(Eq. 4.7)}$$

m controls both the shape and the position of the inbetweens and can be adjusted by user according to the motion he wants to create. Flexible control is provided since m can be of any value. For example, in Figure 4.4, m is set to $\{ \frac{1}{8}, \frac{1}{4}, \frac{1}{2} \}$ for the three inbetweens. This achieves the effect of 'slow in fast out'.



Keyframe 1 Inbetweens Keyframe 2

Figure 4.4: Nonlinear interpolation: jellyfish

In Figure 4.4, the shape and the position changes at the same rate, controlled by m. There are also situations where more complicated movements are desired. For example, an animator may want the shape of the object to change from slow to fast and the position to change from fast to slow. To enable users to control separately over shape and position, we first apply Eq. 4.7 to generate the intermediate frames. Then the elements are translated based on the position parameters specified by animators.

4.3 Feature-Based Stroke Interpolation

The above modification provides more control over shapes and positions of the inbetweens but does not solve all the problems. Figure 4.5 illustrates an example where linear interpolation function fails to generate accurate result. The face silhouette is distorted in the inbetweens. It is because points are evenly taken in parameter domain of two DBSCs as corresponding points for interpolation thus some 'critical' points are not correctly corresponded, e.g. the tip of the nose. Smoothness of lines is critical in animation production and this kind of distortion cannot be accepted in real production.

Keyframe 1 Inbetweens Keyframe 2

Figure 4.5: Linear interpolation: facial silhouette

If a feature point exists in successive keyframes such as the tip of the nose in Figure 4.5, it should also exist in all inbetweens. In this section, we propose a feature-based stroke interpolation method. Feature points are extracted on two strokes and the correspondence is automatically established using a matching algorithm. The interpolation method of two DBSCs is accordingly modified.

4.3.1 Feature Point Extraction

When drawing a stroke, a sequence of points along the trajectory is sampled by input device, such as a digital tablet. To calculate the curvature at a point, the k-cosine technique [34] is applied where the left and the right k^{th} neighbors of a point are used to calculate the angle at the point.

Defining the k-vectors at a point p_i as:

$$\vec{a}_{ik} = (x_i - x_{i-k}, y_i - y_{i-k}) \qquad \text{(Eq. 4.8)}$$
$$\vec{b}_{ik} = (x_i - x_{i+k}, y_i - y_{i+k})$$

the k-cosine at p_i is:

$$\rho_{ik} = \frac{\vec{a}_{ik} \cdot \vec{b}_{ik}}{|\vec{a}_{ik}||\vec{b}_{ik}|} \qquad \text{(Eq. 4.9)}$$

Thus,

$$\theta = cos^{-1}(\rho_{ik}) \qquad \text{(Eq. 4.10)}$$

where $\vec{a}_{ik}$ and $\vec{b}_{ik}$ are respectively the vectors from p_i to its left k^{th} point and to its right k^{th} point, and θ is the angle between $\vec{a}_{ik}$ and $\vec{b}_{ik}$. The sign of the angle is determined by $\vec{a}_{ik} \times \vec{b}_{ik}$. Feature points are defined as points with a small $|\theta|$.

In [33], two immediate neighboring points are used to calculate the angle at a point on a curve, i.e. $k = 1$. It is sensitive to noise of small perturbation which is very common in free-hand drawing hence the calculated angle is not accurate. To overcome the problem, we adopt the method proposed in [37], which integrates the algorithm determining region of support in [34] and the algorithm of corner detection in [25] to choose an appropriate k in order to calculate a more accurate angle value.

For a selected point p_i, the length of the chord joining the points p_{i-k} and p_{i+k} is defined as $L(k)$. $C_l(k)$ and $C_r(k)$ are respectively the contour lengths from p_{i-k} to p_i, and p_i to p_{i+k}. $D_l(k)$ and $D_r(k)$ are respectively the lengths of the chords from p_{i-k} to p_i, and p_i to p_{i+k}. Three conditions are used to determine k:

$$\frac{C_l(k) + C_r(k)}{L(k)} > \frac{C_l(k+1) + C_r(k+1)}{L(k+1)} \qquad \text{(Eq. 4.11)}$$

$$L(k) < L(k+1) \qquad \text{(Eq. 4.12)}$$

$$\begin{cases} \frac{C_l(k) - D_l(k)}{C_l(k)} < 0.1 \\ \frac{C_r(k) - D_r(k)}{C_r(k)} < 0.1 \end{cases} \qquad \text{(Eq. 4.13)}$$

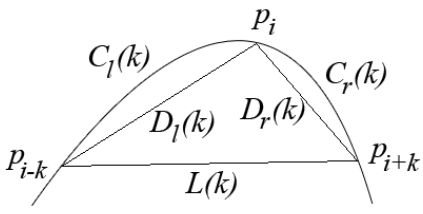
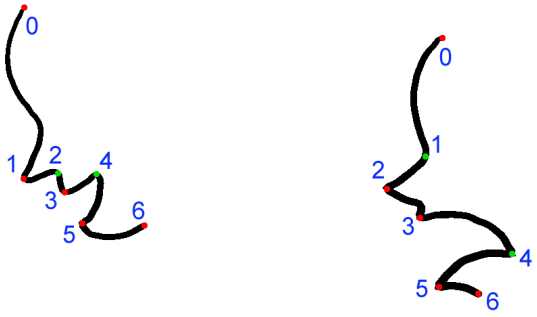


Figure 4.6: Curvature calculation

Start with $k = 1$, repeat $k = k + 1$ until neither Eq. 4.11 nor Eq. 4.12 are satisfied, or Eq. 4.13 is not satisfied. Once k is determined, the angle θ can be calculated. Based on

experiments and investigation of drawings in real production, feature points are selected of a θ less than 140°. By applying the method, feature points can be correctly identified as shown in Figure 4.7. Positive feature points are highlighted in red and negative ones in green. The two ends of a stroke are also treated as feature points. Notice that the number of feature points on two strokes are not necessarily equal.



(a) Stroke S_1 (b) Stroke S_2

Positive feature points are highlighted in red and negative ones in green.

Figure 4.7: Feature point extraction

4.3.2 Feature Point Correspondence

Given two strokes S_1 and S_2, and the ordered feature points on them $P = \{p_0, p_1, p_2, ...p_m\}$ and $Q = \{q_0, q_1, q_2, ...q_n\}$, the best match can be found as the union of the following sets:

- Set A containing all matched pairs: $\{(p_0, q_0), (p_{i_1}, q_{j_1}), (p_{i_2}, q_{j_2})..., (p_m, q_n)\}$
- Set B containing all non-matched elements in P: $\{(p_{k_1}, NULL), (p_{k_2}, NULL)...\}$
- Set C containing all non-matched elements in Q: $\{(NULL, q_{l_1}), (NULL, q_{l_2})...\}$

where

- $0 < i_1 < i_2 < ... < m$;

- $0 < j_1 < j_2 < ... < n$;
- All p_{i_s} and p_{k_t} are unique and their union is P;
- All q_{j_s} and q_{l_t} are unique and their union is Q.

The conditions above guarantee that at most one feature point on a stroke is matched to a counterpart on the other stroke and the order of feature points is preserved in inbetweens. We assume the order of points on the two DBSCs is consistent when a user draws on a tablet thus the two ends are always matched, i.e. (p_0, q_0), (p_m, q_n).

4.3.2.1 Feature Normalization

In [33], angle and parameter costs are used to evaluate a match. Consider the following situation. Given two strokes, among several feature points, each stroke has a point with a sharp turning of opposite directions e.g. $\theta_1 > 0$, $\theta_2 < 0$. Both θ_1 and θ_2 are small angles. The two points look very different and should not be matched. However, since the angles are small, $|\theta_1 - \theta_2|$ results in a low cost and the two points are likely to be matched. To solve the problem, normalized curvature $C(p_i)$ and parameter cost $P(p_i)$ are used, similar to the method in [33]:

$$C(p_i) = \begin{cases} \frac{1}{2\pi} \cdot (\pi - angle) & \text{if } (angle \geq 0) \\ \frac{1}{2\pi} \cdot (-\pi - angle) & \text{if } (angle < 0) \end{cases} \qquad \text{(Eq. 4.14)}$$

$C(p_i)$ is within $[-0.5, 0.5]$ and any $|C(p_i) - C(q_j)|$ is within $[0, 1]$. The parameter cost $P(p_i)$ is within $[0, 1]$ and any $|P(p_i) - P(q_j)|$ is within $[0, 1]$. Thus the curvature cost and the parameter cost have an equal influence in evaluation of matching.

For a single match (p_i, q_j) between two feature points p_i in S_1 and q_j in S_2, the cost function is defined as

$$cost(p_i, q_j) = \begin{cases} |C(p_i) - C(q_j)| + |P(p_i) - P(q_j)| & \text{if } (p_i \neq NULL) \text{ and } (q_j \neq NULL) \\ \lambda_{NULL} & \text{if } (p_i = NULL) \text{ or } (q_j = NULL) \end{cases} \qquad \text{(Eq. 4.15)}$$

where λ_{NULL} is the cost of non-match, which is set to the maximum of 0.5 as a punishment. This drives the matching result to be that as many feature points are corresponded as possible since any non-match will result in a larger cost. The best match minimizes the total cost function:

COST-Total = $\sum$ cost(p_i, q_j) + $\sum$ cost(p_k, $NULL$) + $\sum$ cost($NULL$, q_l)

where (p_i, q_j)$\in$ Set A, (p_k, $NULL$)$\in$ Set B and ($NULL$, q_l)$\in$ Set C.

An efficient recursive algorithm is subsequently applied to solve the optimization problem.

4.3.2.2 Feature point correspondence

The feature matching algorithm is defined as follows, matching two sequences of feature points $P_k : \{p_k, p_{k+1}, ...p_{m-1}\}$ and $Q_l : \{q_l, q_{l+1}, ...q_{n-1}\}$:

```
Match ( { p_i | i ∈ [ k, m-1 ] }, { q_j | ∈ [ l, n-1 ] } )
  If k = m-1 or l = n-1    // found a solution
    If k = m-1
      // Set all remaining elements in list Q_l to non-match
      for a = l to n-1    // inclusive
        Set (NULL, q_a)
    Endif
    If l = n-1
      // Set all remaining elements in list P_k to non-match
      for a = k to m-1    // inclusive
        Set (p_a, NULL)
    Endif
    Compute COST-Total    // calculate the total cost of current matching
```

```
    If COST-Total < COST-BestMatch        // update the best match
        Set COST-BestMatch = COST-Total
    Endif
  Otherwise    // to find a match for p_k
    Set (p_k, NULL)    // case 1: p_k is matched to NULL
    Call Match ( { p_i | i ∈ [ k+1, m-1 ] }, { q_j | j ∈ [ l, n-1 ] } )
    for a = l to n-1    // inclusive
      Set (p_k, q_a)    // case 2: p_k is matched to q_a (a = l, l+1, ..., n-1)
      for b = l to a-1    // inclusive
        Set (NULL, q_b)
      Compute COST-Total    // calculate current total cost
      If COST-Total < COST-BestMatch    // if current total cost is smaller
      // than the best match, go on with iteration; otherwise, abort current match
        Call Match ( { p_i | i ∈ [ k+1, m-1 ] }, { q_j | j ∈ [ a+1, n-1 ] } )
      Endif
  Endif
```

For Figure 4.7, the best match is:

$$BestMatch = \{(p_0, q_0), (NULL, q_1), (p_1, q_2), (p_2, NULL), (p_3, q_3), (p_4, q_4), (p_5, q_5), (p_6, q_6)\}$$

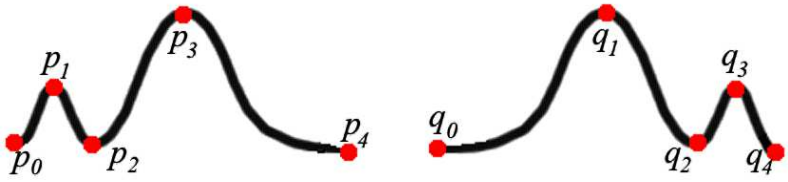
The minimum cost $COST(S_1, S_2)$ is 1.48.

4.3.2.3 Ambiguity Of matching

While the matching algorithm is completely automatic and yields accurate results in most situations, ambiguity may occur in some cases such as in Figure 4.8. Naturally, there can be two matches:

$$BestMatch_1 = \{(p_0, q_0), (p_1, q_1), (p_2, q_2), (p_3, q_3), (p_4, q_4)\}$$

$$BestMatch_2 = \{(p_0, q_0), (p_1, NULL), (p_2, NULL), (p_3, q_1), (NULL, q_2), (NULL, q_3), (p_4, q_4)\}$$



(a) Stroke S_1 (b) Stroke S_2

Figure 4.8: Ambiguity of matching

Both results can be obtained with different λ_{NULL} in the algorithm and both may be considered correct mathematically. We believe it is more appropriate to let users choose the one they desire in case of ambiguity.

4.3.3 Stroke Interpolation Based on Feature Points

Now the interpolation function is modified accordingly. Suppose two DBSCs and their feature points are given, i.e. the control points: P_i $(i = 1, 2, ..., m)$ and Q_j $(j = 1, 2, ..., n)$ and the feature points: P_{i_k} and Q_{i_k} $(k = 1, 2, ...l)$. For the data points of the first DBSC, during interpolation, we use chord length for the knot vector, i.e.

$$\begin{cases} u_0 = 0 \\ u_i = u_{i-1} + |P_{i+1} - P_i| \end{cases} \quad \text{(Eq. 4.16)}$$

For the data points on the second DBSC, the knot vector is determined by adjusting the chord length through scaling so that these feature points are corresponded in parametric domain.

$$\begin{cases} v_0 = 0 \\ v_j = v_{j-1} + |Q_{j+1} - Q_j| \cdot s_j \end{cases} \quad \text{(Eq. 4.17)}$$

$$s_j = \frac{\sum_{t=i_k}^{i_{k+1}} |P_{t+1} - P_t|}{\sum_{t=j_k}^{j_{k+1}} |Q_{t+1} - Q_t|} \qquad \text{(Eq. 4.18)}$$

for $j = j_k + 1, j_k + 2, ..., j_{k+1}$ $(k = 1, 2, ..., l)$.

By applying the method above, the inbetween result of Figure 4.7 is illustrated in Figure 4.9. Another two examples are shown in Figure 4.10 and 4.11. In comparison with the previous method in Section 4.2.3, it is obvious that feature-based stroke interpolation reduces distortion and shapes of strokes are better preserved.

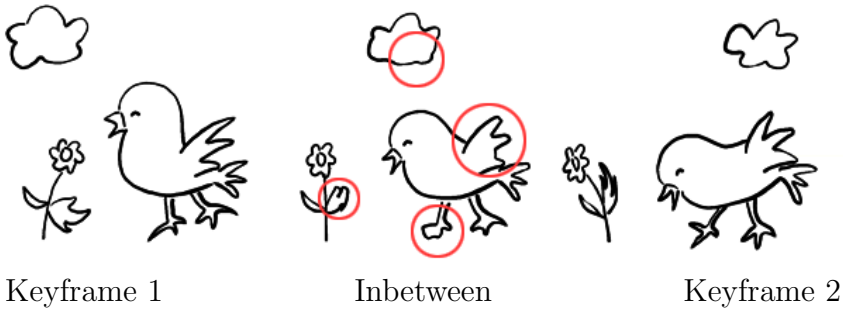
Keyframe 1 Inbetweens Keyframe 2

Figure 4.9: Feature-based stroke interpolation: facial silhouette

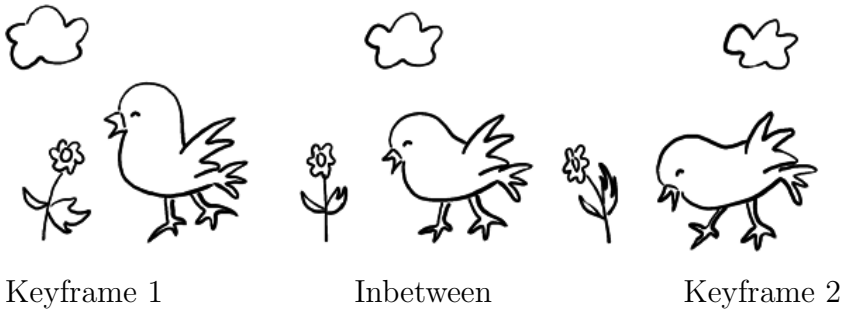
4.4 Motion-Enhanced Inbetween Generation

4.4.1 Handling Single Object

The method proposed in Section 4.3 preserves features of single strokes. Another problem occurs when a character has big motion, e.g. rotation. For example, the arms are shortened in inbetweens in Figure 4.11. Another example in Figure 4.12 shows a comparison between the techniques we proposed here. Although feature-based stroke interpolation preserves feature of the strokes in Figure 4.12(b), the overall shape is distorted since no



(a) Linear interpolation (Main distortion circled)



(b) Feature-based stroke interpolation

Figure 4.10: Inbetween generation results: chicken

global motion information is considered in interpolation. Figure 4.13 also illustrates a similar situation where the left arm and the sword are shortened or distorted.

Both linear and non-linear interpolation are based on the spacial position of keyframes without regards to the basic nature of movements especially for character animation. Now we treat the inbetween problem as not just a geometric problem, but a motion between two or more keyframes. We advance our proposed technique in Section 3.4 here, in which the whole character can be divided into several approximately rigid components and thus the transformation of each segment is a rigid motion through translation, rotation and scaling. The global motion minimization technique can be adapted. By aligning the pose parameters of the first keyframe to those of the second one, the two keyframes are registered to minimize the global motion. Figure 4.14(a) shows the result by applying global

Keyframe 1 Inbetweens Keyframe 2

(a) Linear interpolation (Main distortion circled)

Keyframe 1 Inbetweens Keyframe 2

(b) Feature-based stroke interpolation

Figure 4.11: Inbetween generation results: fireman

Keyframe 1 Inbetweens Keyframe 2

(a) Linear interpolation

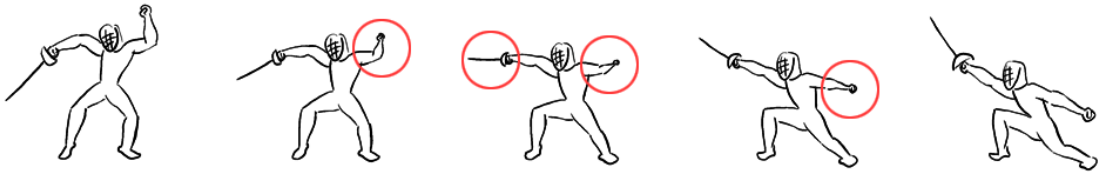
Keyframe 1 Inbetweens Keyframe 2

(b) Feature-based stroke interpolation

Figure 4.12: Comparison of inbetween generation techniques: goldfish I

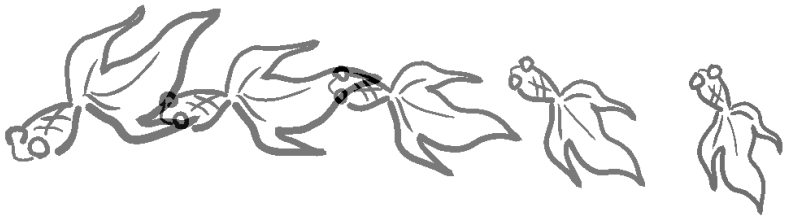
motion estimation to a single object in addition to feature-based stroke interpolation.

The method is also critical when user specifies a motion path along which the object

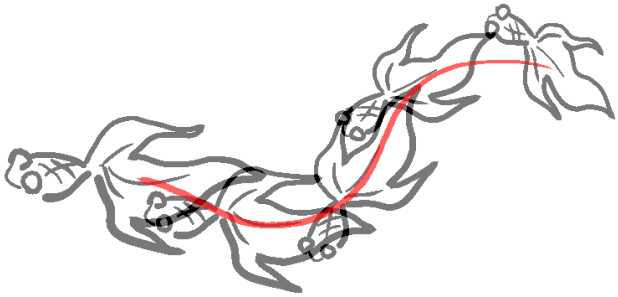


Keyframe 1 Inbetweens Keyframe 2
(Main distortion circled)

Figure 4.13: Linear inbetween generation: fencing



Keyframe 1 Inbetweens Keyframe 2
(a) Motion-enhanced interpolation based on Figure 4.12(b)



(b) Path-driven inbetween generation based on (a)

Figure 4.14: Comparison of inbetween generation techniques: goldfish II

moves. To obtain a rotating effect rather than simple translation along the path, the global motion of the given two keyframes is estimated and a sequence of inbetweens are generated with minimized distortion. The tangent at any point on the path can be computed. Thus the inbetweens can be transformed so that the global pose is aligned with the tangent of the points on the path. A path can be represented by any spline curve. In our approach, we simply use DBSC for consistence. If the user provides a sequence of

points, we can obtain a path through interpolation or approximation of DBSC. Figure 4.14(b) shows the final sequence.

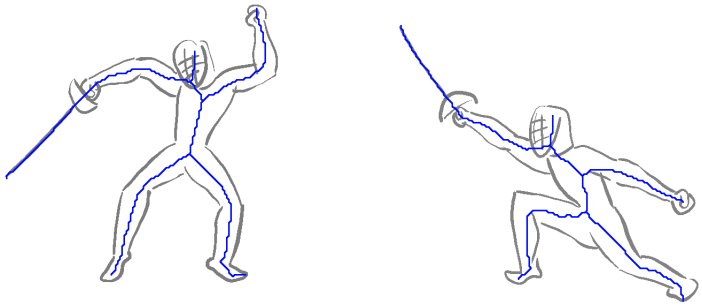
4.4.2 Hierarchical Structure

As discussed in Section 3.4, the motion of a character is usually more complicated. Various components will undergo different motion thus motion estimation should be applied to each component. No matter how a character moves, its structure normally remains unchanged.

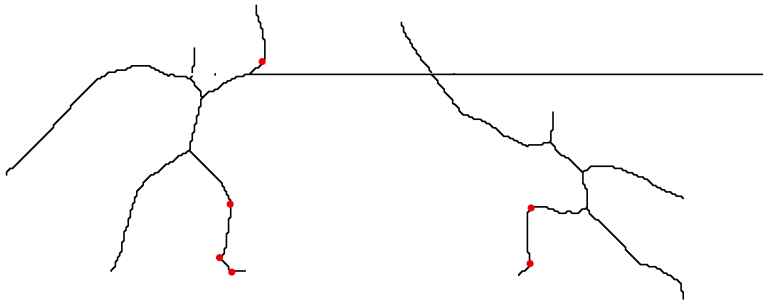
We advance the approach proposed in Section 3.4. Firstly the skeletons of the character in the two keyframes are extracted and corresponded. The process is shown in Figure 4.15(a) using the keyframes in Figure 4.13.

Now we need to decompose the character into components based on the skeleton. The raster-based approach in Section 3.4.2.2 segments the character by pixels. In vector representations, a stroke is an important element. A drawing is composed of strokes and inbetween generation is based on stroke-to-stroke correspondence, thus a stroke should be preserved throughout a sequence. We hence consider all data points on a stroke as a whole in order not to break a stroke into different segments. To evaluate the distance of a stroke S_i to a certain branch B_j, the average distance $D_{i,j}$ of all data points on S_i to B_j is calculated. S_i is associated with the branch B_k to which $D_{i,k}$ is the smallest. In this way, each stroke will be associated with a branch of the skeleton. The graph matching algorithm in Section 3.4.3 is adopted to match the skeletons of two keyframes based on length of branches and neighboring relationship between them. Since each branch is related to a component, the components in the two keyframes are matched accordingly.

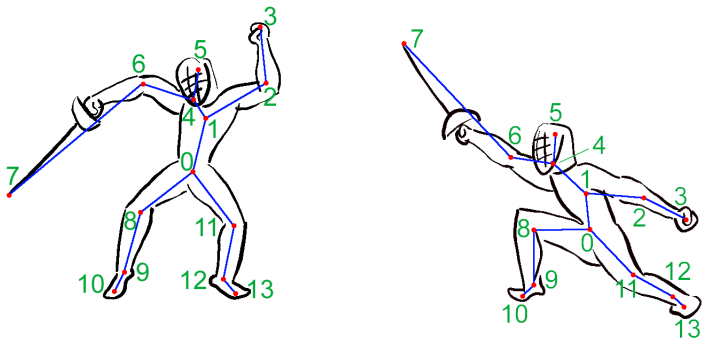
A bending on a branch usually indicates different global motion inside the shape e.g. the bending legs in Figure 4.15(a). In this case the branch as well as the corresponding



(a) Skeleton extraction



(b) Feature point extraction



(c) Matched and optimized skeletons

Figure 4.15: Skeleton correspondence

component should be further segmented into two. The method in Section 4.3.1 is applied to extract feature points, as shown in Figure 4.15(b). If no corresponding feature point

is found on its matched branch, one will be automatically inserted at the corresponding position so that the two skeletons can be matched. Finally a branch can be simplified into a straight line.

In current approach, distance calculation is used for decomposition without any topological information. Some strokes might be associated with a branch which is not structurally correct. Besides, all feature points may not be identified e.g. the elbow of the right arm in Figure 4.15(b). The arm bends smoothly thus it has no feature point in both keyframes. Minor user-interaction is incorporated here while more robust approach towards automation is being investigated. Figure 4.15(c) shows the final result of two matched skeletons by manually inserting one more feature points at the elbow of the right arm.

The junctions of the skeleton indicate how various parts are related. By manually specifying the root node, a hierarchical structure is established. The corresponding hierarchical structure of the skeleton in Figure 4.15(b) is shown in Figure 4.16, where the junction node at the pelvis, i.e. *Node* 0, is specified as the root node. The structure consists of several levels. The root node, *Node* 0, is at level 0, the highest level. Those directly connected to it through a branch are in the next level, so on so forth. With the hierarchical structure, any transformation applied to a node can be propagated down to all its descendants in lower levels.

4.4.3 Kinematics-Enhanced Interpolation

To generate smooth motion according to the structure of the character, we incorporate forward kinematics, which is commonly used in 3D animation, into our proposed technique of global motion estimation. Starting from the root node of the skeleton, its global motion from keyframe 1 to keyframe 2 is estimated. Given only one node, only translation can be estimated. The transformation is applied to the node in keyframe 1 so that

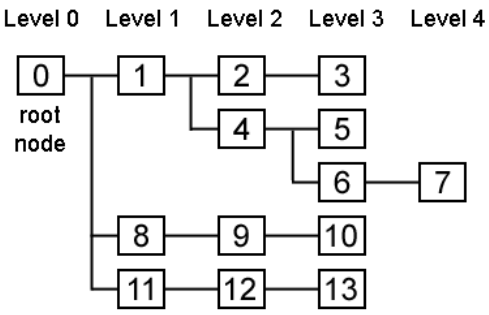


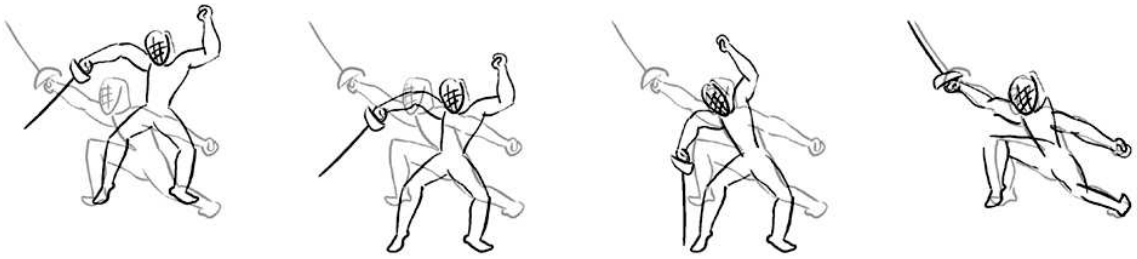
Figure 4.16: Hierarchical structure

it is aligned with that in keyframe 2. The transformation is then applied to all descendants of *Node* 0 in the hierarchical structure. A transformation of a node also drives the motion of the branch linking the node and its exclusive parent node. A branch is related to a number of strokes as described in Section 4.4.2. Therefore, those related strokes are applied with the same transformation. The alignment of *Node* 0 is illustrated in Figure 4.17(b). For the next node i.e. *Node* 1, the same procedure applies, except that not only translation but also rotation and scaling are estimated, as shown in Figure 4.17(c). The estimation goes from higher levels to lower ones until every node in keyframe 1 is aligned to that in keyframe 2. Figure 4.17(d) shows the final result of alignment.

Now interpolation is applied to the relocated keyframes to generate inbetweens. To restore the motion of various components, inverse procedure is applied, from leaf nodes, i.e. those in the lowest level, to the root node.

The final inbetween results are shown in Figure 4.18. Since the global motion is minimized before interpolation, the problem of distortion is solved and the shape is well retained.

Another example is illustrated in Figure 4.19. Notice that the two skeletons extracted from the two keyframes by thinning are not consistent. As depicted in the circle in Figure 4.19(a)(ii), in this area, keyframe 1 has two split nodes while keyframe 2 has only one



(a) (b) (c) (d)

(a) Original keyframes

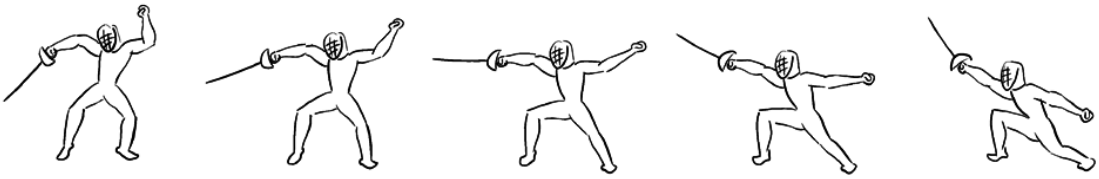
(b) Registering *Node* 0

(c) Registering *Node* 1

(d) Registering all nodes

Relocated keyframe 1 in darker strokes; keyframe 2 in lighter.

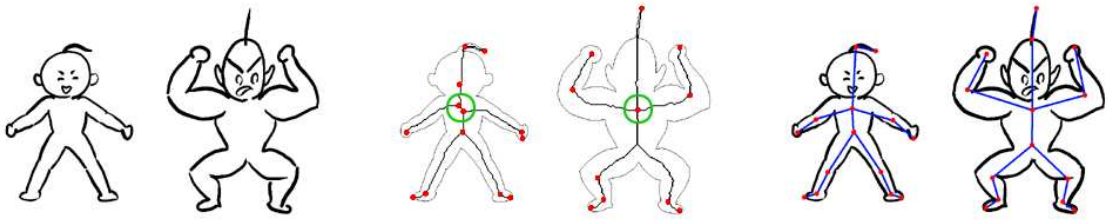
Figure 4.17: Global motion estimation



Keyframe 1 Inbetweens Keyframe 2

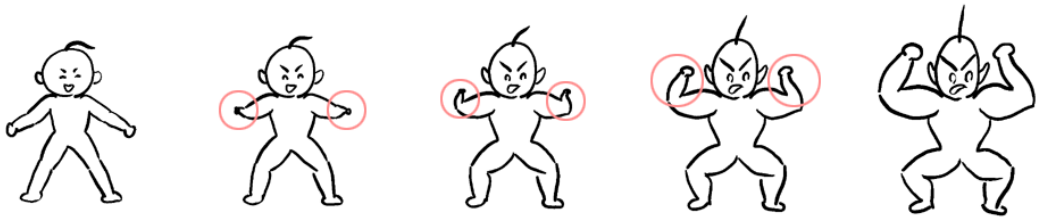
Figure 4.18: Motion-enhanced inbetween generation: fencing

node. In this case, the two nodes in keyframe 1 will be merged automatically since they are connected and very close to each other, which can be detected. Thus both keyframes will have consistent optimized skeletons in Figure 4.19(a)(iii) and the same hierarchical structure.



(i) Keyframes (ii) Skeletons with feature points (iii) Optimized skeletons

(a) Global motion estimation



Keyframe 1 Inbetweens Keyframe 2

(b) Linear inbetween generation (Main distortion circled)

Keyframe 1 Inbetweens Keyframe 2

(c) Motion-enhanced inbetween generation

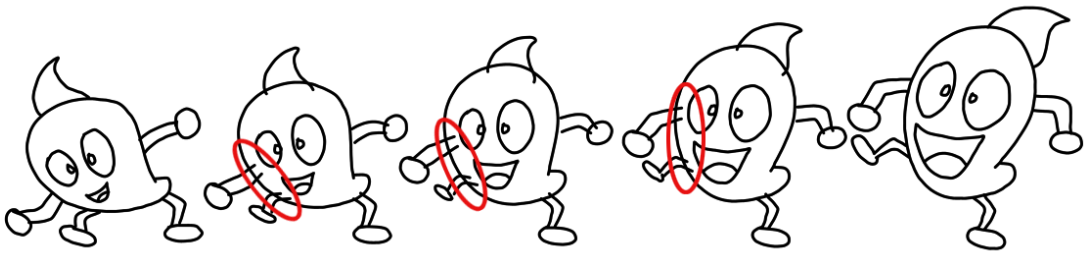
Figure 4.19: Inbetween generation results: muscle boy

4.5 Junction Preservation in Inbetween Generation

In keyframe drawings, strokes intersect one and another, which often forms regions to represent meaningful parts of objects or characters. In Figure 4.20, the arms and the legs are always connected to the contour of the body in the keyframes. These touching points

are notified as 'junctions' here. In general, it is a type of intersection of two DBSCs in which at least one intersection point is an end point of a DBSC. If this junction point occurs in successive keyframes, it should also be preserved among all inbetweens to maintain shape features. However, since no constraint is placed in the interpolation function, the points may not connect in inbetweens as depicted in the circles in Figure 4.20.

Preserving junction points is important, not only to maintain the real human structure, but also to facilitate subsequent coloring process. A region must be closed to apply coloring in its interior. Any gaps on the boundary will result in the color 'leaking' out of the region. Although small gaps can be filtered by existing gap closing algorithms, it is more effective and efficient to avoid them at the time of generation than to fix them afterwards. In this section, we propose an efficient method to compute junctions and preserve them in inbetweens.



Keyframe 1 Inbetweens Keyframe 2

Figure 4.20: Interpolation without junction preservation: bean (Main artefacts circled)

4.5.1 Intersection Detection

Since a junction is a type of intersection of two DBSCs, we will work on the general detection of intersections. In [87], several curve intersection algorithms are compared, including the well known Bézier subdivision algorithm, Bez-Sub, described in [47]. Bez-Sub relies on the convex hull property of Bézier curves and on the de Casteljau algorithm

for sub-dividing Bézier curves. A rational Bézier curve, with non-negative weights, lies within the convex hull of its control points. Thus the algorithm compares the convex hulls of two curves first. Only when they overlap will they be subdivided and the new convex hulls be checked for overlap. Each iteration rejects parts of the curves which do not contain intersection points.

Another algorithm Int-Sub proposed by [11] is similar to Bez-Sub. A curve is pre-processed to determine its vertical and horizontal tangents and the curve is divided into intervals that have horizontal or vertical tangents only at endpoints of the intervals. It is based on the observation that within the intervals, a rectangle defined by any two points on the curve, i.e. data points, completely bounds the curve between those two points. The algorithm uses a bounding rectangle instead of convex hull, and subdivision is applied by evaluating the (x, y) coordinates of the midpoint of the interval and defining the two resulting rectangles.

DBSC is a mathematical model whose center curve is a B-spline curve. The difference is that DBSC incorporates radius at each point, which composes a disk. Due to this property, the intersection of two DBSCs can be rephrased as:

Two DBSCs intersect where control disks on the two DBSCs overlap.

It is easy to convert a B-spline to a piecewise Bézier curve. Similarly, the mathematic model of DBSC supports efficient conversion to a piecewise disk Bézier curve. For a DBSC with control disks $\{(P_0, r_0), (P_1, r_1), ..., (P_{n-1}, r_{n-1})\}$, the control disks of the converted piecewise cubic disk Bézier curves are:

$$\{Q_{0,0}, Q_{0,1}, Q_{0,2}, Q_{1,0}, Q_{1,1}, Q_{1,2}, ...Q_{n-2,0}, Q_{n-2,1}, Q_{n-2,2}, Q_{n-1,0}\}$$

$\{Q_{i,0}, Q_{i,1}, Q_{i,2}, Q_{i+1,0}\}$ $(i = 0, 1, ..., n-2)$ are the four control disks of a cubic disk Bézier curve. Given two piecewise Bézier curves, the algorithms described in [87] can be applied to compute intersections fast. The problem thus can be reduced to computing intersections of two disk Bézier curves.

Here we propose an efficient method to compute intersections of two DBSCs. Firstly, we examine the convex hull property of a disk Bézier curve. As shown in Figure 4.21, a disk Bézier curve is a Bézier curve with radius at each point. Thus it has control disks instead of control points. The Bézier convex hull is illustrated in Figure 4.21. For a disk Bézier curve, the radii of the control points must be taken into account. Suppose the i^{th} disk has the largest radius. Now we expand all other control disks so that they have the same radii as the i^{th} disk. The external tangential line of two successive control disks can be found which does not intersect with the convex hull of the center Bézier curve. The four tangential lines together with the four control disks form a region. It is clear that the disk Bézier curve lies within the region. To simplify computation, we compute the bounding square of each control disk and then the convex hull of all the vertices of the squares. The convex hull encloses the disk Bézier curve as well. With this powerful property, now we can adapt the method in [47] which subdivides two curves and checks the overlapping of the convex hulls for each pair.

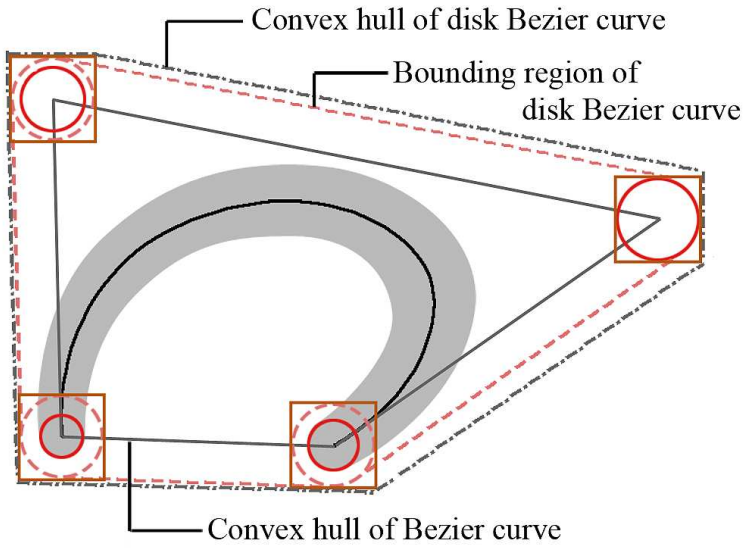


Figure 4.21: Convex hull property of a disk Bézier curve

In [47], iterations of subdivision repeat until a curve segment approximates a line segment within a specified tolerance. If two such 'straight line' segments overlap, the intersection point is accepted as an intersection of the two curves. For DBSC, overlap of data disks needs to be detected in order to find intersections, as shown in Figure 4.22. In such cases, the center curves may not intersect at all. To obtain accurate results, we sample in parametric domain to compute the disks on a disk Bézier curve with a high sampling rate based on the length of the curve. By linking the control points using line segments, we calculate the total length l of all segments as the approximate length of the disk Bézier curve. Pre-setting a sampling step d, the number of disks, n, to be sampled is:

$$n = inf(length/step) + 1 \qquad \text{(Eq. 4.19)}$$

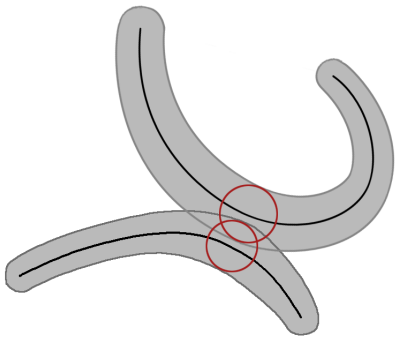


Figure 4.22: Intersections of two disk Bézier curves

The sampling knots in parametric domain are calculated as follows:

$$knot_i = \frac{i}{n-1}(i = 0, 1, ..., n-1) \qquad \text{(Eq. 4.20)}$$

The more accurate result is desired, the smaller the sampling step should be set, which may result in higher computational cost. An image is discretized in pixel level

when being displayed at a certain resolution. Thus a sampling step of 0.5 is sufficient to obtain accurate results when the image is displayed.

Now we compute the distance between each pair of sampled data disks on two disk Bézier curves. For a disk $D_{1,i}$ on the curve S_1 and a Disk $D_{2,j}$ on the curve S_2, the distance between $D_{1,i}$ and $D_{2,j}$ is calculated as follows:

$$d_{i,j} = |C_{1,i} - C_{2,j}| - r_{1,i} - r_{2,j} \qquad \text{(Eq. 4.21)}$$

where $C_{1,i}$, $C_{2,j}$ are the centers of $D_{1,i}$ and $D_{2,j}$ respectively; $r_{1,i}$ and $r_{2,j}$ are the radii of $D_{1,i}$ and $D_{2,j}$ respectively. If $d_{i,j} \leq 0$, S_1 and S_2 overlap at the two disks. A local minimum is found to be an intersection of S_1 and S_2. Figure 4.23 shows the intersections of two DBSCs computed by our method.

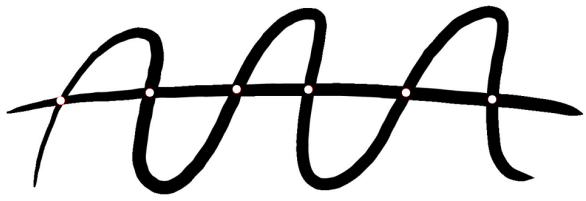


Figure 4.23: Intersections of two DBSCs

By applying the method, we can obtain the parametric value and the point coordinates of each intersection, which will be used in the following process. Notice that it can also be utilized to detect self-intersection of a single DBSC.

4.5.2 Junction Preservation

Once junctions are detected of the same stroke pairs in both starting and ending keyframes, it should be preserved in inbetweens.

Assume strokes S_1 and S_2 form a junction in both keyframes. One point is an end point P of S_1 and the other is a non-end point Q of S_2. We then check whether the two strokes intersect in inbetweens. Two situations need to be handled:

1. S_1 and S_2 intersect at point R, where a small artefact occurs. In this case, we remove the artefact.

2. S_1 and S_2 do not intersect, which means a gap occurs between P and Q in inbetweens. In this case, we change the position of P.

In both situations, S_2 remains unchanged so that its shape is preserved. Figure 4.24 illustrates the situations.

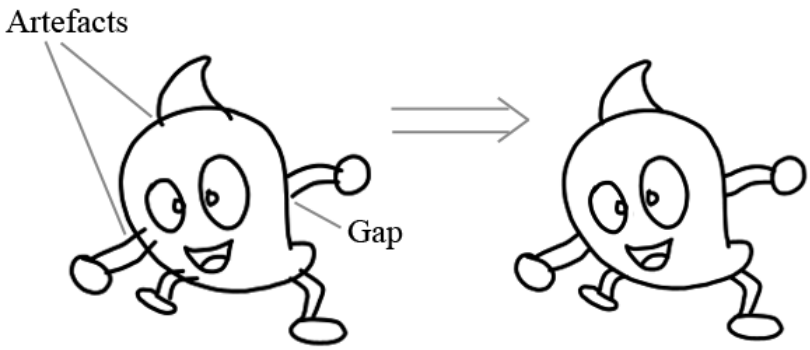
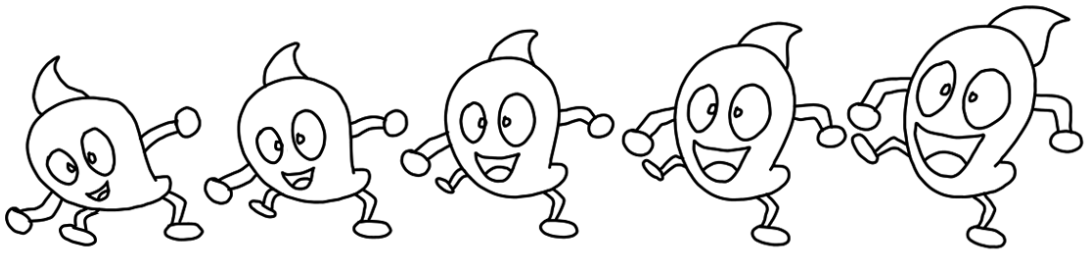


Figure 4.24: Junction preservation

Now for case 2, the correct position of P in inbetweens needs to determined. From the intersection detection method, we obtain the parametric value of the intersection point Q on S_2, which is t_1 in keyframe 1 and t_2 in keyframe 2. Suppose n inbtweens are generated. Based on the interpolation function, P_k, the position of P in inbetween I^k $(k = 0, 1, ..., n-1)$ can be computed. The parametric values of Q in I^k is computed as follows:

$$t_k = t_1 \cdot (1 - \frac{k+1}{n+1}) + t_2 \cdot \frac{k+1}{n+1} \qquad \text{(Eq. 4.22)}$$

From the parametric value, we can compute Q_k, the position of point Q in inbetween I^k. To preserve the junction, P_k must be equal to Q_k. Therefore, we set $P_k = Q_k$ $(k = 0, 1, ..., n-1)$ so that the two points always connect in the inbetweens. The result is shown in Figure 4.25, where the artefacts in Figure 4.20 are removed by preserving the junction points. Another two examples are shown in Figure 4.26 and 4.27. It is obvious that by applying our proposed approach, undesired gaps or artefacts are eliminated and inbetweens of better quality are generated.

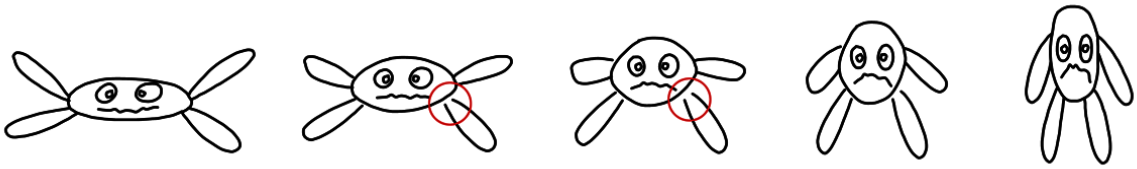


Keyframe 1 Inbetweens Keyframe 2

Figure 4.25: Inbetween generation with junction preservation

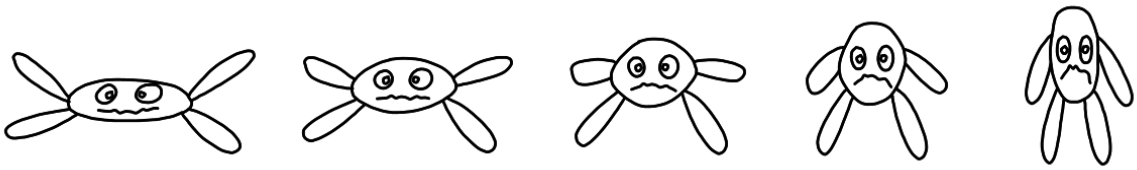
4.6 Summary

In this chapter, our research focuses on the area of vector-based inbetween generation. To solve distortion problem in the inbetweens, various information is extracted from keyframes and utilized in inbetween generation. We first propose an approach of feature-based stroke interpolation where feature points on two strokes are automatically extracted and corresponded in interpolation. Secondly we propose a novel technique of motion-enhanced inbetween generation, where global motion of a character's components is estimated and interpolated. Finally we introduce a method to detect junction points and preserve them in inbetweens. By applying these techniques, shapes are preserved, distortion is eliminated and smoother inbetween result is achieved.



Keyframe 1 Inbetweens Keyframe 2

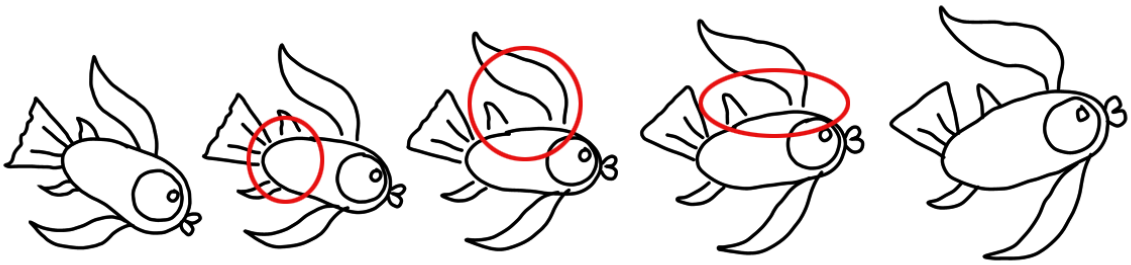
(a) Inbetween generation without junction preservation (Main gaps circled)



Keyframe 1 Inbetweens Keyframe 2

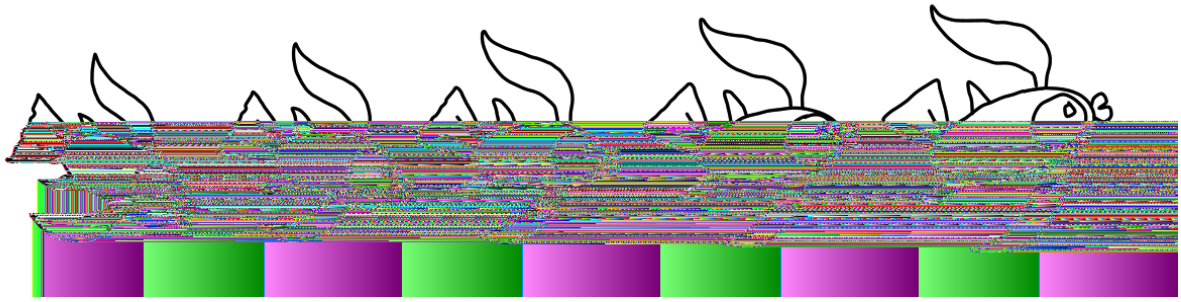
(b) Inbetween generation with junction preservation

Figure 4.26: Inbetween generation results: sea-creature



Keyframe 1 Inbetweens Keyframe 2

(a) Inbetween generation without junction preservation
(Main gaps circled)



Keyframe 1 Inbetweens Keyframe 2

(b) Inbetween generation with junction preservation

Figure 4.27: Inbetween generation results: fish

Chapter 5

Vectorization of Raster Line Drawings in Cartoons

5.1 Introduction

Despite the digitization of many procedures in cartoon production, it is still common for animators to draw frames with pencil and paper rather than on a digital tablet. In computer-assisted cartoon production, vectorization is commonly required for the raster line drawing images so that further applications can be performed easily, such as scaling and auto-painting. Some attractive techniques of auto-painting are based on shape matching, which is more suitable for processing vector-based graphics [41]. Efficiency of storage is another advantage of vector representation. Hundreds of frames need to be created to produce a cartoon film and a vectorized representation greatly reduces the storage. The result of vectorization is affected by various factors such as the scanning process and the non-uniqueness of vectorization [89]. Our aim is to obtain a vectorized image as similar as possible to the original drawing while using relatively few data.

Rosenfeld [13, 14] introduced a number of algorithms for raster-to-vector conversion. Other works have also been done to vectorize a raster image using different approaches of approximation, such as polygonal approximation [82, 39, 43], curve fitting [10, 38, 32, 63, 64], or skeletonization [41, 89]. However, these conventional methods obtain one-pixel

wide skeleton but neglect a significant characteristic of artists's drawings i.e. the different width of strokes which embodies the expressions of artists, especially in the categories such as animations of traditional Chinese style drawing as shown in Figure 5.1, a piece by a famous Chinese cartoonist, Feng Zikai. Artistic information will be lost. To capture the characteristic, regions depicting strokes should be represented.

Figure 5.1: Traditional Chinese style cartoon drawing

There are mainly two kinds of techniques of 2D shape representation i.e. boundary-based e.g. chain coding, polygonal approximation and spline interpolation, and region-based e.g. Medial Axis Transformation, *MAT*. In our approach, we will use the novel representation of freeform shapes, disk B-spline curve (*DBSC*), as introduced in Section 4.2. It describes a region as well as its center curve thus provides useful information, which is exploited to preserve the style of original drawings. It provides more information than the boundary-based methods and uses less data compared with MAT.

According to the properties of DBSC as described in Section 4.2.1, a smooth shape can be represented by a DBSC. A line drawing of cartoon frame usually comprises a number of strokes and can be represented by a set of DBSCs. Therefore, our main task to vectorize a line drawing by DBSC is to segment the drawing into strokes, extracting

the data disks, i.e. the center points of the strokes as data points of DBSC and their radii.

There are a few existing methods that may achieve the goal. MAT obtains the centers and radii of *maximal disks* [31]. Thus it seems to be a suitable method for our application. However, though quite a few methods of MAT have been proposed [15, 49, 24], for a raster image which is discretized and quantified, a disconnected or even an empty skeleton may be obtained by MAT and the skeleton may not be one pixel wide. This will bring difficulty in segmenting strokes. Besides, most methods have high computational cost.

Another solution is thinning method. A typical thinning algorithm iteratively peels off boundary pixels according to a set of rules until a one-pixel wide, connected subset of the original shape remains [91, 51, 21]. It generates a single-pixel-wide skeleton, which is required for tracing and segmenting. The main disadvantage is that it may not obtain accurate center points.

In this chapter we propose a novel approach taking advantages of both MAT and thinning. The details will be elucidated in the following sections.

5.2 Data Disk Extraction

Our vectorization method consists of three processes: skeletonization, segmentation and adjustment. In skeletonization, a raster line drawing is thinned into a one-pixel-wide skeleton image. After tracing, we can segment the image into strokes, which consist of ordered points. Adjustment is applied to centerize the points of the skeleton. The details of the method are explained in the following sections.

5.2.1 Skeletonization and Stroke Segmentation

After an original drawing is scanned into the computer, it is converted to binary. Then SUSAN thinning algorithm [51] is applied to obtain the skeleton, followed by a smoothing

process to remove the artifacts. The process is similar to that proposed in Section 3.4.1. The difference is that in Section 3.4.1 thinning is applied to the whole shape within the character boundary to obtain the main structure, neglecting any details. Here we will apply thinning to the original images to capture all details.

We use an example of a single region in Figure 5.2 to illustrate our method. A general cartoon drawing (such as those shown in Section 5.6) usually consists of a set of disconnected strokes and can be solved by iteratively searching for unprocessed separate strokes. Figure 5.2 shows the original shape in shadow and the thinning result. The original shape is thinned into a one-pixel-wide skeleton. However, it can be noticed that some points are not on the center curves, e.g. those within the circle. Generally SUSAN algorithm does not guarantee the results to be the center lines.

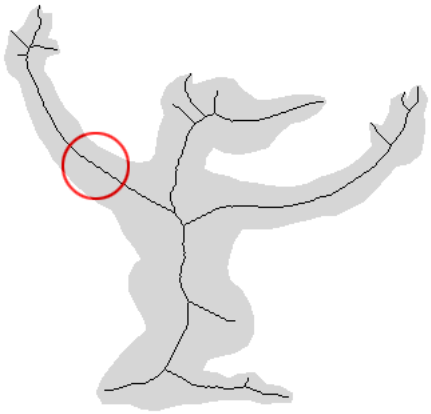


Figure 5.2: Skeletonization

Now that the skeleton is one pixel wide and connected, it can be traced so as to segment the original shape into strokes using the method in Section 3.4.2.1. The skeleton in Figure 5.2 consists of 26 strokes in total.

5.2.2 Point Centerization

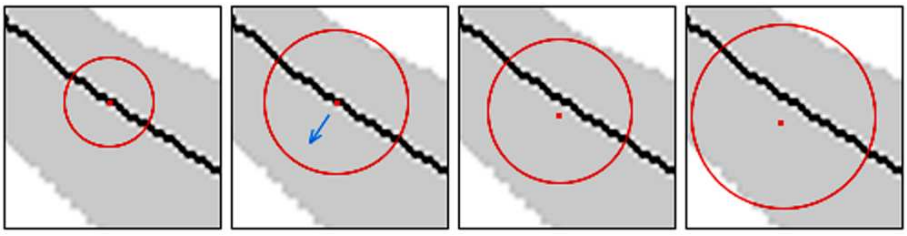
Next, the points will be adjusted to the center curve of the stroke based on the definition of maximum disk in MAT [31]. We define a circle with its center at the current point

and iteratively calculate the centroid of all white (background) pixels contained in the circle, which results in the following situations:

(i) If there is no white pixel in the circle yet, the radius of the circle will be increased with the center unchanged (Figure 5.3(a)). The centroid of white pixels within the circle will be recalculated.

(ii) If the distance from the calculated centroid to the circle center is beyond a pre-set threshold based on experiments, it infers that there exists only one contact of the circle with the shape boundary (Figure 5.3(b)) i.e. currently the circle center is not on the center curve. Therefore the circle center moves a step in the direction from the centroid to current circle center (Figure 5.3(c)) and the iteration continues.

(iii) If the distance from the centroid to the circle center is within the threshold, it infers that there is more than one contact of the circle with the shape boundary. This means current circle is the maximal disk and the circle center is right on the center curve. In this case, the iteration stops. The original point will be adjusted to current circle center and the radius of the circle will also be recorded (Figure 5.3(d)).



(a) (b) (c) (d)

Figure 5.3: Sample point centerization

Note that as a DBSC needs only a small number of data disks to represent a smooth shape as depicted in Section 4.2. Hence only a proportion of points need to be adjusted and used as data points. This greatly reduces computational time.

Due to the rounding computation of discretized raster image, deadlocks may occur as shown in Figure 5.4. After some iterations, the point is currently adjusted to location a in the defined discrete circle. The centroid of white pixels contained in the circle is thus at a'. According to the rules we specified, the radius is unchanged and the circle center a is adjusted to b. The corresponding centroid of white pixels is then at b'. The iteration goes on and the point is subsequently adjusted to c (the corresponding centroid is at c'), d (the corresponding centroid is at d') and a again, which forms a deadlock. In this situation, notice that the circle center is adjusted repeatedly without any change in the length of the radius. This implies that there is always some contact with the boundary after each adjustment i.e. the point is already on the center curve thus the iteration stops.

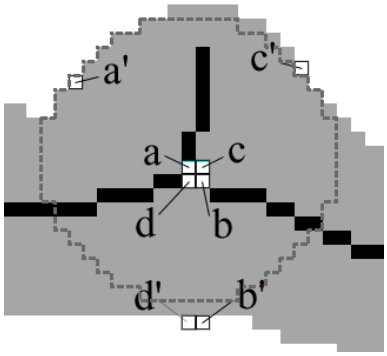


Figure 5.4: Deadlock in centerization

Since thinning already generates a skeleton composed of approximate 'center' points, the number of iterations is mainly affected by the width of the strokes i.e. the radii of the maximum disks. Assuming the starting radius, the increment of the radius and the movement of the circle center all as 1 pixel in our procedure, the average number of iterations is 20.6 in Figure 5.2, where the maximum and the minimum radius among all circles is 41 pixels and 0 respectively. For images with thinner strokes, the iteration ends quickly and the computation is more efficient. Additional examples will be given in Section 5.6.

5.3 Sampling Rate

As stated earlier, the minimum number of sample points that a DBSC needs to represent a smooth stroke is 4. For better efficiency, different sampling rates should be applied to strokes of different length.

A variable sampling rate is adopted according to the length of strokes for compromise between the quality of vectorization and the quantity of data. We define the sampling rate $1/n$, where n is the number of disks between two subsequent sampled disks. An initial sampling rate is pre-defined based on experiments. If the number of sampled disks on current stroke is less than 4 which is required for DBSC interpolation, the rate is increased until there are sufficient disks sampled. In this way, higher sampling rate is applied to shorter strokes and lower rate for longer strokes. Figure 5.5 shows the sampled data disks. In total, 139 disks are sampled.

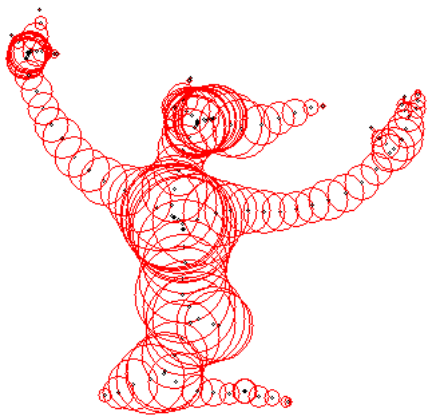


Figure 5.5: Sampled data disks

5.4 Reconstruction

Now the interpolation function of DBSC can be applied to the data disks to get the control disks of DBSC, as illustrated in Figure 5.6. The consecutive control points are

connected by black lines. The corresponding stroke is represented by a DBSC. To examine the result, a reconstructed image is generated by drawing all the DBSCs onto a canvas. Figure 5.7 shows the original shape (in shadow) and the reconstructed image (outline only) for comparison. The *peak signal to noise ratio,* ***psnr***, is used to evaluate the difference between two images. *psnr* is based on the root mean square error, *rmse*, which is the square root of the sum of all the differences in corresponding pixel intensities between two images. Assuming an image resolution of $m \times n$ pixels,

$$rmse = \sqrt{\frac{1}{m \times n} \sum_{j=1}^{m} \sum_{k=1}^{n} (i_{j,k} - i'_{j,k})^2} \qquad \text{(Eq. 5.1)}$$

where $i_{j,k}$ and $i'_{j,k}$ are the intensities of $(j,k)^{th}$ pixel in the two images respectively. With a p-bit pixel intensity, the *psnr* is defined as follows:

$$psnr = 20log(\frac{2^p - 1}{rmse})\text{dB} \qquad \text{(Eq. 5.2)}$$

The *psnr* for Figure 5.7 is 40.62 dB.

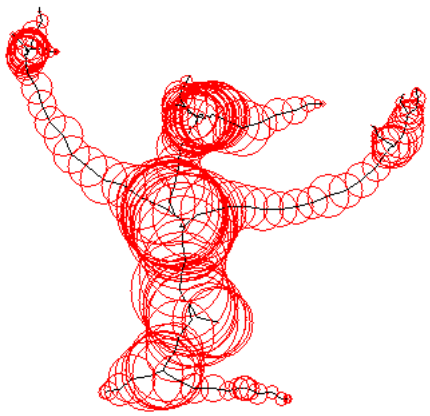


Figure 5.6: Control disks by interpolating data disks

The result shows the overall shape of the reconstructed image is quite close to the original one. Some details are not captured such as the top of the left hand and the contour around the left side of the waist. The former is due to the thinning algorithm

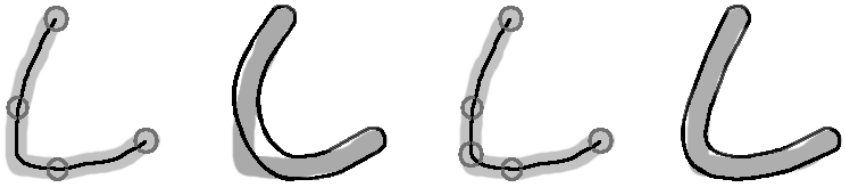
Figure 5.7: Superimposed original image and vectorized image
(original image in shadow and vectorized image in outline)

which results in a missing stroke of a finger. This could be solved by further improving the thinning algorithm in the future so that it captures all necessary details. The latter results from a low sampling rate of the stroke and can be solved by increasing the rate at the cost of more data.

5.5 Feature-Enhanced Sampling

A fixed sampling rate for a single stroke may also cause problem. Figures 5.8 gives an example where (a) shows the sampled disks on an original stroke and (b) shows the comparison between the original stroke (in shadow) and its reconstruction (outline only). The actual length of the stroke is small thus only four data disks are sampled, which a characteristic corner of the stroke is missed. It can be solved by increasing the sampling rate but will result in redundant sampling for the smoother parts on the stroke. An efficient solution is to sample feature points which indicate major turnings of the stroke. The curvature at each point of the skeleton can be calculated using the technique of the k-cosine measure for corner detection proposed by [34]. The method used in Section 4.3.1 is adopted which is a direct operation on a sequence of points and has low computational cost.

By the definition in Section 4.3.1, feature points are defined as points with a small θ. Based on experiments, feature points are selected to be the points where $|\theta| \leq 90\,^\circ$ in our approach. Once the feature points are sampled, the original shape can be better captured as shown in Figures 5.8(c) and (d). A small number of sampling points are used here for clearer view. More points should be sampled to achieve accurate result.



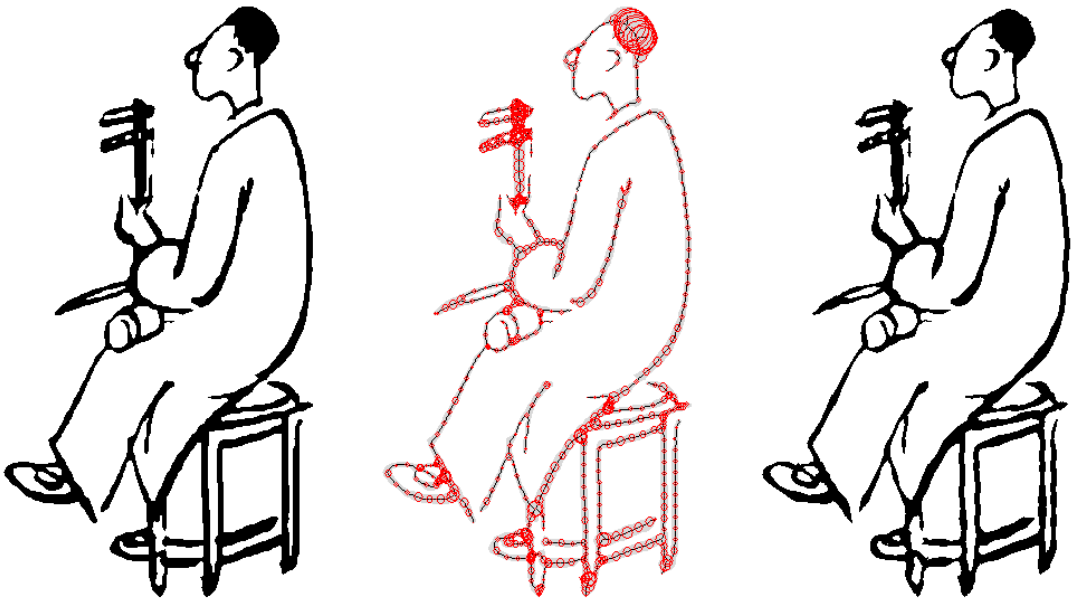
(a) (b) (c) (d)

Figure 5.8: Feature-enhanced sampling

5.6 Results and Discussion

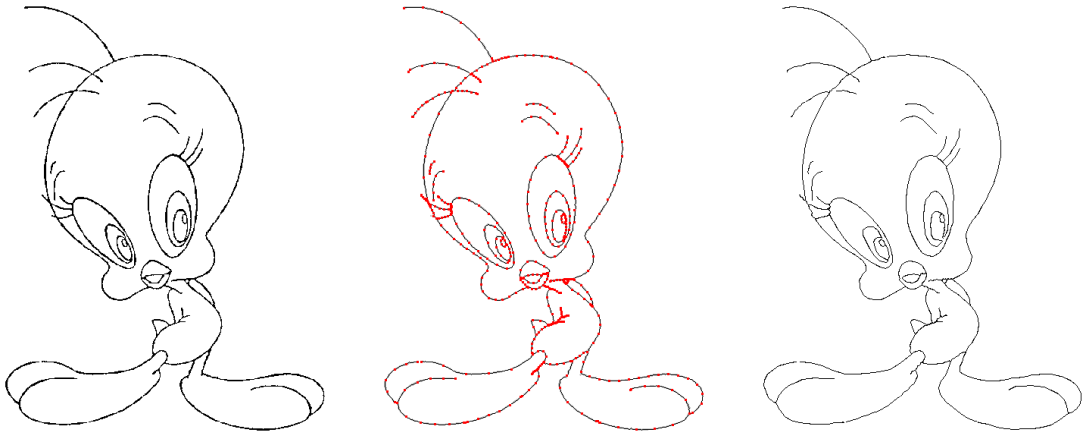
The algorithm has been tested on some images shown in Figure 5.9, a traditional Chinese style drawing, Figure 5.10, a traditional cartoon frame of Tweety in *Space Jam*, Figure 5.11, a piece of Chinese calligraphy by a Chinese calligrapher, Zhang Zhi, and Figure 5.12, a frame of a Japanese Anime in real production. In each example from left right, it shows (a) the original image, (b) the sampled disks, and (c) the vectorized image. For Figure 5.9 to Figure 5.11, the total execution time from thinning to reconstruction takes about 5-8 seconds, depending on the size and the complexity of the drawing. The vectorization of Figure 5.12 takes more than a minute due to its tremendous size of 3190×2873 and complexity of drawing, which is required for high-quality production. The process is fully automatic. When in some cases only a one pixel wide skeleton is required e.g. Figure 5.10, all the radii can be simply set to 0.5.

Table 5.1 lists the dataset of the original images and the vectorized versions. The original images are in TIFF format, which is uncompressed and provides superior quality



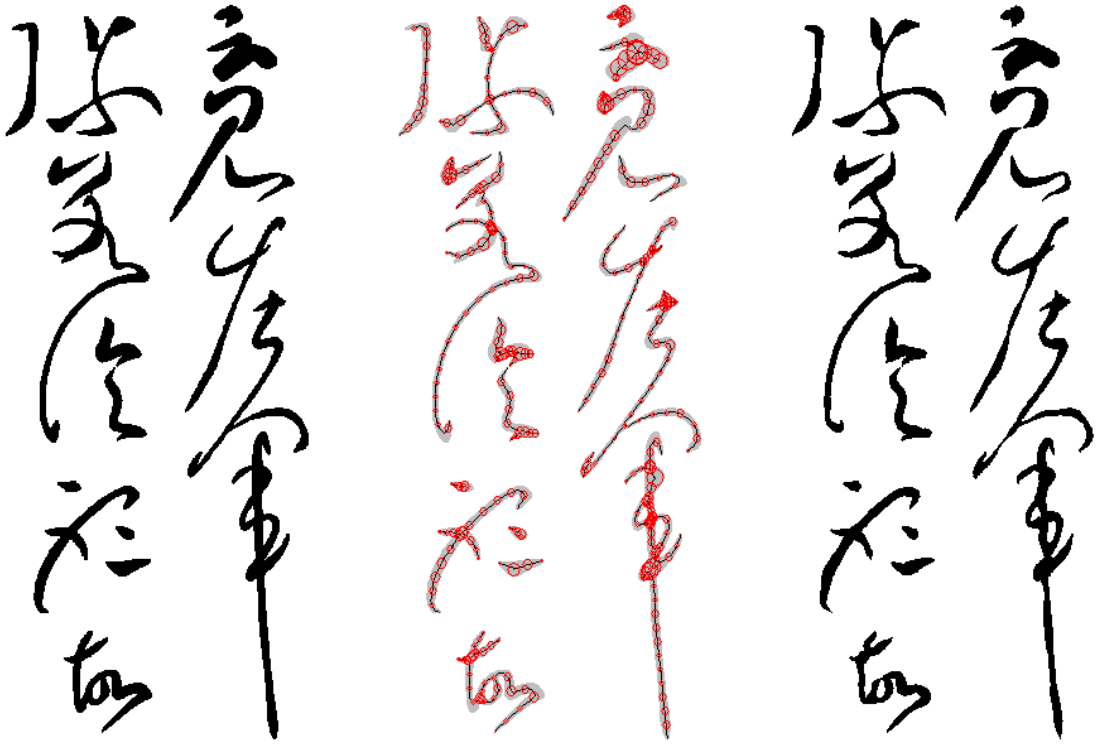
(a) Original image (b) Sampled data disks (c) Vectorized image

Figure 5.9: Vectorization result: Traditional Chinese style cartoon drawing



(a) Original image (b) Sampled data disks (c) Vectorized image

Figure 5.10: Vectorization result: Tweety
(courtesy of Warner Bros.)



(a) Original image (b) Sampled data disks (c) Vectorized image

Figure 5.11: Vectorization result: Chinese calligraphy

(a) Original image (b) Sampled data disks (c) Vectorized image

Figure 5.12: Vectorization result: Tsukasa
(courtesy of Anime International Co., Inc.)

for real production. For vectorized images, only the center coordinates and the radius of each sampled disks need to be recorded in text files, which greatly reduces the storage. Since the radii are all set to 0.5 in Figure 5.10, the dataset can be further condensed. The average number of iterations when centerizing the sampled points and the *psnr* between the original and the vectorized images are also listed.

Table 5.1: Dataset of vectorization

	No. of Strokes	No. of data disks	Average no. of iterations
Figure 5.9	118	668	4.3
Figure 5.10	84	567	1.3
Figure 5.11	79	444	3.6
Figure 5.12	396	1160	2.2

	Tiff image size (pixels)	Tiff image size (kb)	Text file size (kb)	*psnr* (dB)
Figure 5.9	339×600	601	12	37.5
Figure 5.10	500×639	940	8	41.1
Figure 5.11	261×607	469	8	35.4
Figure 5.11	3190×2873	26,875	31	26.2

From the results, it is obvious that the vectorized images are quite close to the original ones judging by human eyes while the data required are much less. It proves that the method is effective, especially in vectorization of large images such as cartoon drawings in practical production. In our method, the resulting vector form only contains binary information of the pixel instead of greyscale. This is adequate because in real animation production, constant black lines are used for inking the outlines. There is no change of the color. However, DBSC also supports intensity and color change in a single stroke in specific applications.

Currently *psnr* is used in our method to evaluate the dissimilarity of the original raster image and the reconstructed image rendered from the vector representation. A higher

psnr generally indicates better quality of vectorization. *psnr* is based on pixel intensity hence is rather an image-based evaluation without any topological evaluation. However, preserving topology is important in vectorization. It will cause problem e.g. color leaking if a closed region becomes open after vectorization. More appropriate evaluation should be adopted to evaluate the topology. Improvement of the proposed vectorization method can also be further researched so that topology in the original drawings is preserved.

5.7 Summary

In this chapter, we presented an automatic approach of line drawing vectorization using a new model of freeform shape representation, DBSC. A method based on thinning and MAT is applied to obtain data disks of the strokes in original raster images. Firstly thinning is applied to get an approximate skeleton. Individual strokes can be segmented by tracing the skeleton. Then the points on the skeleton are adjusted to the center curve of the corresponding stroke according to the definition of MAT and their radii can meanwhile be calculated. This is computationally efficient. Experiment results show the effectiveness of the method in both retaining the style of original drawings and reducing size of dataset.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

This thesis investigates computer-assisted inbetween generation in 2D animation production. The ultimate target of our research is to reduce workload of manual inbetween drawing hence increase productivity and creativity of animators. Most existing methods are based on either pixel intensity of raster images or mathematical functions of vector representation, but neglect information in the drawing itself, e.g. shapes, structures and motion of characters. This commonly results in shape distortions and unsmooth motion. Our research aims to solve the problems and generate smooth and realistic inbetweens. Our work is carried out in both image-based and vector-based areas. In image-based area, we aim to solve the problem of inaccurate correspondence of keyframes due to pixel-level matching. In vector-based area, we focus on solving the problems resulting from common linear interpolation.

Various techniques are proposed in this thesis in order to extract various information from keyframes and incorporate it in the process of inbetween generation. Our main contributions are summarized as follows:

In image-based area:

- A novel idea and technique is proposed to handle global and local motion of characters respectively. A method of keyframe relocation is devised to minimize global

motion inspired from methods in traditional animation production. It automatically estimates global motion of a character and minimize it between keyframes. Together with Modified Feature-based Matching Algorithm (MFBA), which is adopted to estimate local changes, the technique achieves automation and handles a wide variety of inbetweening problems, for both gray scale images and line drawings.

- For a character with complicated structure and motion, a novel idea is proposed to handle different motion of various components based on segmentation. A novel image-based character segmentation scheme is devised based on the skeleton which is automatically extracted from the character. Motion estimation is then applied to each component. The results prove effectiveness of our methods in achieving accurate matching of keyframes and smoother inbetweens.

In vector-based area:

- Various critical information in keyframes is investigated, including feature points on a stroke, hierarchical structure and motion of a character, and junction points between strokes. The information is extracted from keyframes and employed into the interpolation process. Distortion is eliminated and smoother sequence of animation is achieved.

- A hierarchical approach of character segmentation in vector-representation is proposed. A hierarchical structure of the character is established based on the extracted skeleton. Global motion of the character's various components is estimated and kinematic interpolation is applied to the entire structure. The method preserves shapes in the movement and eliminates the shorting problem in rotation.

- A method to detect junction points in keyframes and preserve them through inbetweens is presented. A junction point is a type of intersection. Accordingly,

intersection problem of disk B-spline curves (DBSC), the vector representation of strokes we utilized, is investigated. The convex hull property of DBSC is proved. With this powerful property, intersections thus junctions of DBSCs can be detected. The approach preserves relationship between strokes and closed regions.

In image-to-vector area:

- Based on the requirement in the industry to convert raster images to vector representation, an efficient vectorization technique is proposed based on thinning algorithm and Medial Axis Transformation (MAT). It traces and samples original images as strokes. Data disks of the strokes are computed and interpolated to generate the vectorized image represented by DBSCs. Convincing results prove effectiveness of the method in both efficiency of data reduction and resemblance between originals and vectorized images. The technique can be used as the base of further process in vector-based area.

The approaches proposed in this thesis have been implemented and incorporated in a software which is used by renowned Japanese anime production company to produce latest animes and the quality is generally accepted by professional animators.

6.2 Future Work

There is still much room to be researched and improved in current approaches we proposed.

Currently, we apply segmentation by pixels in image-based methods and by strokes in vector-based methods. Either pixels or strokes are low-level elements in a frame, which mainly provide information of the image intensity or the way of drawing. They do not provide information of the structure of a character. Pixels of a region may be segmented

to adjacent component due to pure distance calculation from pixels to skeleton branches. This may lead to incorrect segmentation and the integrity of regions or component is not guaranteed. Examining a character, a hierarchical structure can be established in this way: a character consists of components; a component consists of regions; and finally a region consists of pixels or strokes. If segmentation is performed on a higher level, such as by regions or group of regions, pixels or strokes contained in the same region will be considered as a whole. This will lead to more accurate and meaningful segmentation. In addition, subsequent coloring process is applied to regions. It is appropriate to detect regions and use them as the basic elements for processing.

In image-base area, a region can be traced by searching pixels along boundaries. In vector-based area, the results of intersection detection of two DBSCs described in Section 4.5.1 can be utilized. From the intersections computed, we can trace along the DBSC since we already have the information of strokes in mathematical representation. Once a character is successfully divided into regions, it will facilitate subsequent processing such as coloring and may provide more flexible control and manipulation over a drawn character.

Further applications in cartoon making may be extended based on our method of vectorization. With the flexible representation of DBSC, operations like triangulation, visualization and texture mapping can be applied to the points of the strokes straightforwardly. Some attributes can be defined on the strokes, such as a scalar field, vector field, which may be utilized in stylized rendering. Generally images of superior quality can be generated with raster imagery, but with vector imagery, textures and patterns are usually difficult to replicate. However, DBSC supports great flexibility in texture due to its intrinsic properties. This may lead to further research and applications.

In current method, the correspondence between strokes in keyframes is established by the order of drawing, which somehow constrains the conventional way artists draw.

This problem is more critical in the vectorization process. After vectorization, the correspondence between strokes in keyframes is not established, thus interpolation cannot be applied directly. Establishing correspondence automatically between pairs of DBSCs in two images is a difficult task [12, 17]. If a stroke intersects with a few other strokes in the original drawing, it may be represented by several DBSCs after vectorization. More advanced technique of stroke matching between keyframes needs to be investigated in future research.

The approaches proposed in this thesis handle translation and 2D rotation. However, there are more complicated movements in traditional animation, such as rotation in a 3D-like way. These situations are extremely difficult to be handled by computer. As Catmull [23] pointed out, the main problem is due to the fact that keyframe drawings are barely 2D projections of 3D objects with the original spatial information lost. Therefore introducing 3D information is the most possible way to solve the problem.

One of the solutions is to construct a 3D model to provide 3D information. Considering the traditional inbetween drawing, an animator does have a 3D appearance of the character in his mind when he draws. However, employing full 3D input models with *Non-photorealistic rendering* (NPR) techniques may not be practical in 2D animation. The 3D modelling phase is complicated and usually done manually, which requires a lot of time and labor. In addition, 2D animation involves subtle artistic changes, exaggerated actions as well as specific drawing style of artists. These characteristics can hardly be captured by a rigid 3D model without complicated manipulation. Therefore, an accurate 3D model is not suitable for 2D animation.

Another idea is to use an approximate 3D model which may be constructed at lower cost. In traditional production, animators usually draw rough primitives first to approximately outline parts of a character and later polish them with all detailed features, as shown in Figure 6.1. An approximate model allows reduction and simplification to the benefit of higher level processing.

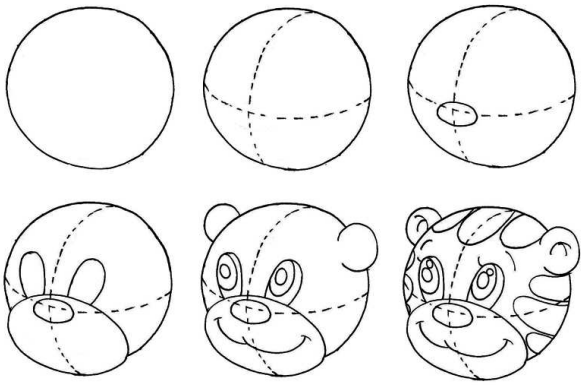


Figure 6.1: Rough to detailed drawing

To build such a 3D model, some freeform modelling techniques such as [78, 59] may be exploited. It may also help by using *master frames*. *Master frames* serve as a reference for drawing and they portray a character from characteristic viewpoints offering most details as illustrated in Figure 6.2. It is proper and feasible to construct the model based on them.

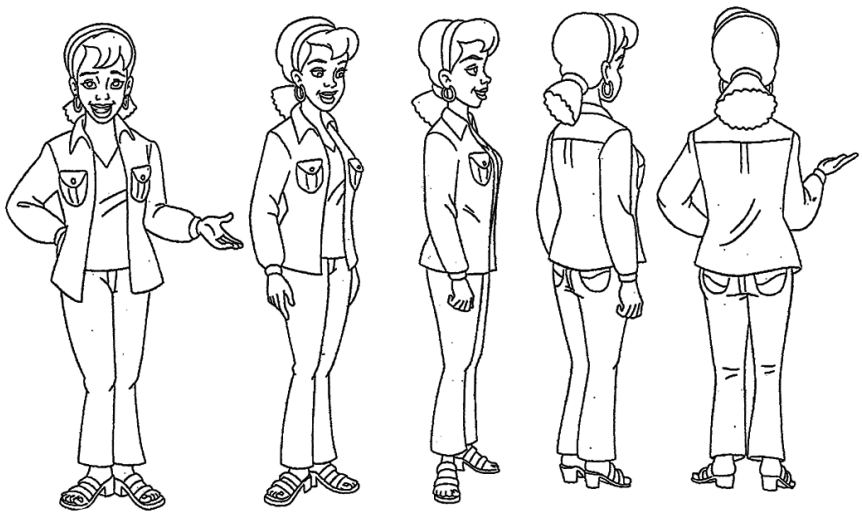


Figure 6.2: Master frames
(by courtesy of NELVANA Limited)

In all, introducing 3D information to inbetween generation is a significant but tough

task which is worthy of being researched and may lead to breakthrough in the problem of inbetween generation and possibly other aspects of computer-assisted cel animation.

References

[1] *http://plasticanimationpaper.dk/.*

[2] *http://www.animationstand.com/.*

[3] *http://www.bauhaussoftware.com/products mirage studio.php.*

[4] *http://www.cambridgeanimation.com/products/animo.htm.*

[5] *http://www.celaction.com/.*

[6] *http://www.macromedia.com/software/flash/.*

[7] *http://www.retas.com/.*

[8] *http://www.toonboom.com/products/.*

[9] *http://www.toonz.com/.*

[10] Barsky B A, Bartels R H, and Beatty J C. *An introduction to Splines for use in computer graphics and geometric modeling.* Morgan Kaufmann, 1987.

[11] Koparkar P A and Mudur S P. A new class of algorithms for the processing of parametric curves. In *Comput.-Aided Des.*, volume 15, pages 41–45, 1983.

[12] Kort A. Computer aided inbetweening. In *ACM Press*, pages 125–132, 2002.

[13] Rosenfeld A. Algorithms for image/vector conversion. In *Computer Graphics*, number 12, pages 135–139, 1978.

[14] Rosenfeld A and Kak A C. *Digital image processing.* Academic Press, New York, 1987.

[15] Telea A and Van Wijk J J. An augmented fast marching method for computing skeletons and ceterlines. In *Joint EUROGRAPHICS - IEEE TCVG Symposium on Visualization*, 2002.

[16] A. Agarwala and S. NV. Snaketoonz : A semiautomatic approach to creating cel animation from video. In *Proceedings of NPAR 2002*, 2002.

[17] Di Ruberto C. Attributed skeletal graphs for shape modelling and matching. In *12th International Conference on Image Analysis and Processing*, pages 554– 559, 2003.

[18] Gonzalez R C and Woods R E. *Digital image processing.* Prentice Hall, second edition, 2002.

[19] Gotsman C and Surazhsky V. Guaranteed intersection-free polygon morphing. *Computers & Graphics*, 25(1):67–75, 2001.

[20] Litwinowicz P C. Inkwell: a $2\frac{1}{2}D$ animation system. In *Computer Graphics / Proc. SIGGRAPH*, pages 113–122, 1991.

[21] Brunner D and Brunnett G. Mesh segmentation using the object skeleton graph. In *Proceedings of the 7th IASTED conference on Computer graphics and imaging*, 2004.

[22] Van den Bergh J, Di Fiore F, Claes J, and Van Reeth F. Interactively morphing irregularly shaped images employing subdivision techniques. In *1st Ibero-American Symposium on Computer Graphics 2002 (SIACG 2002)*, pages 315–321, 2002.

[23] Catmull E. The problems of computer-assisted animation. In *Computer Graphics, ACM*, volume 12(3), pages 348–353, 1978.

[24] Remy E and Thiel E. Medial axis for chamfer distances: computing look-up tables and neighbourhoods in 2D or 3D. In *Pattern Recognition Letters*, volume 23(6), pages 649–661, 2002.

[25] Arrebola F, Bandera A, Camacho P, , and Sandoval F. Corner detection by means of contour local vectors. *Electronics Letters*, (38(14)):699C701, 2002.

[26] Di Fiore F and Van Reeth F. Employing approximate 3D models to enrich traditional computer assisted animation. In *Proceedings of Computer Animation*, 2002.

[27] Di Fiore F and Van Reeth F. Mimicing 3D transformations of emotional stylised animation with minimal 2D input. In *ACM Press*, pages 21–28, 2003.

[28] Di Fiore F, Schaeken P, Elens K, and Van Reeth F. Automatic in-betweening in computer assisted animation by exploiting 2.5D modeling techniques. In *Proceedings of Computer Animation*, pages 192–200, 2001.

[29] Van Reeth F. Integrating $2\frac{1}{2} - D$ computer animation techniques for supporting traditional animation. In *Computer Animation '96*, pages 118–125, 1996.

[30] Wolberg G. Image morphing: a survey. *The Visual Computer*, 14(8/9):360–372, 1998.

[31] Blum H. A transfromation for extracting new descriptors of shape. In *Models for the Perception of Speech and Visual Form*. Cambridge MA: MIT Press, 1967.

[32] Chang H H and Yan H. Vectorization of hand-drawn image using piecewise cubic Bezier curves fitting. In *Pattern Recognition*, volume 31, pages 1747–1755, 1998.

[33] Johan H, Koiso Y, and Nishita T. Morphing using curves and shape interpolation techniques. In *Proceedings of Pacific Graphics 2000, The Eighth Pacific Conference on Computer Graphics and Applications*, pages 348–358, 2000.

[34] Teh C H and Chin R T. A scale-independent dominant point detection algorithm. In *IEEE trans Computer Vision & Pattern Recogonition*, pages 229–234, 1988.

[35] Halas J and Manvell R. *The technique of film animation*. Hastings House, New York, 1968.

[36] Lu J, Seah H S, and Tian F. Computer-assisted cel animation: post-processing after inbetweening. In *International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia, GRAPHITE*, pages 13–20, 2003.

[37] Qiu J, Seah H S, Tian F, Wu Z, and Chen Q. Feature- and region-based auto painting for 2D animation. *The Visual Computer*, 21(11):928–944, October 2005.

[38] Schneider P J. An algorithm for automatically fitting digitized curves. In *Graphics Gems*, pages 612–625, San Diego, London, 1990. Academic Press.

[39] Sklanky J and Gonzales V. Fast polygonal approximation of digitized curves. In *Pattern Recognition*, number 12, pages 327–331, 1980.

[40] Weng J, Huang T S, and Ahuja N. *Motion and structure from image sequences*. Springer-Verlag, 1993.

[41] Zou J J and Yan H. Vectorization of cartoon drawings. In *ACM International Conference Proceeding Series from Pan-Sydney workshop on Visualisation*, volume 2, pages 77–78, 2000.

[42] Fekete JD, Bizouarn E, Cournarie E, Galas T, and Taillefer F. TicTacToon: a paperless system for professional 2D animation. In *Computer Graphics Proceedings*, Annual Conference Series, pages 79–90, 1995.

[43] Wall K and Danielsson P. A fast sequential method for polygonal approximation of digitized curves. In *Computer Vision, Graphics and Image Processing*, 28, pages 220–227, 1984.

[44] Piegl L and Tiller W. *The NURBS book*. Springer-Verlag, 1995.

[45] Qun L and Jon G. Rokne: disk Bézier curves. In *Computer Aided Geometric Design*, 15(7), pages 721–737, 1998.

[46] Alexa M, Cohen-Or D, and Levin D. As-rigid-as-possible shape interpolation. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 157–164. ACM Press/Addison-Wesley Publishing Co., 2000.

[47] Lane J M and Riesenfeld R. A theoretical development for the computer generation and display of piecewise polynomial surfaces. In *IEEE Trans. RAMI*, volume 2, pages 35–46, 1980.

[48] Owen M and Willis P. Modelling and interpolating cartoon characters. In *IEEE*, 1994.

[49] Rumpf M and Telea A. A continuous skeletonization method based on level sets. In *Proceedings VisSym'02, Barcelona*, 2002.

[50] Shapira M and Rappoport A. Shape blending using the star-skeleton representation. In *Computer Graphics and Applications, IEEE*, volume 15, pages 44–50, 1995.

[51] Smith S M. Edge thinning used in the SUSAN edge detector. *Internal Technical Report TR95SMS5, Defence Research Agency, Chobham Lane, Chertsey, Surrey, UK*, 1995.

[52] Sonka M, Hlavac V, and Boyle R. *Image precessing, analysis and machine vision.* International Thomson Computer Press, 1993.

[53] Thalmann N M and Thalmann D. *Computer animation: theory and practice.* Springer-Verlag, 1985.

[54] Thorne M, Burke D, and van de Panne M. Motion doodles: an interface for sketching character motion. In *ACM Transactions on Graphics*, volume 23, pages 424–431, 2004.

[55] Xie M. Feature matching and affine transformation for 2D cel animation. *Visual Computer*, (12):419–428, 1995.

[56] D. Macrini, A. Shokoufandeh, S. Dickinson, K. Siddiqi, and S. Zucker. View-based 3-D object recognition using shock graphs. In *16th International Conference on Pattern Recognition*, volume 3, pages 24–28, 2002.

[57] Arad N, Dyn N, Reisfeld D, and Yeshurun Y. Image warping by radial basis functions: applications to facial expressions. *CVGIP: Graph Models Image Processing*, (56):161–172, 1994.

[58] Burtnyk N and Wein M. Interactive skeleton techniques for enhancing motion dynamics in key frame animation. In *Communications of the ACM 19*, pages 564–569, 1976.

[59] Karpenko O, Hughes J F, and Raskar R. Free-form sketching with variational implicit surfaces. In *Computer Graphics Forum*, volume 21, September 2002.

[60] Gao P and Sederberg T W. A work minimization approach to image morphing. *Visual Comput*, (14), 1998.

[61] Litwinowicz P and Williams L. Animating images with drawings. *Comput Graph*, pages 409–412, 1994.

[62] Rademacher P. View-dependent geometry. In *Proceedings of SIGGRAPH 1981*, pages 439–446, 1999.

[63] Zhang S Q, Li L, and Seah H S. Vectorization of digital images using algebraic curves. In *Computers & Graphics*, volume 232, pages 91–101, 1998.

[64] Zhang S Q, Li L, and Seah H S. Fine-tuning in vectorization using algebraic curves. In *Computers & Graphics*, volume 23, pages 269–276, 1999.

[65] Bartel RH and Hardock RT. Curve-to-curve associations in spline-based inbetweening and sweeping. In *Proceedings of the 1989 ACM SIGGRAPH conference*, pages 167–174, 1989.

[66] D. F. Rogers. *Procedural elements for computer graphics*. McGraw-Hill, America, 1998.

[67] Lee S, Chwa K-Y, Hahn J, and Shin S Y. Image morphing using deformable surfaces. In *Proceedings of Computer Animation '94*, pages 31–99, 1994.

[68] Lee S, Chwa K-Y, Hahn J, and Shin S Y. Image morphing using deformation techniques. *J Visualization Comput Anim*, (7):3–23, 1996.

[69] Madeira J S, Stork A, and Grob M H. An approach to computer-supported cartooning. *Visual Computer*, (12:1C17), 1996.

[70] Seah H S and Chua B C. A skeletal line-joining algorithm. In *Insight Through Computer Graphics, Proceedings of the Computer Graphics International (CGI 94)*, pages 62–73, 1994.

[71] Seah H S and Tian F. Computer-assisted coloring by matching line drawings. *The Visual Computer*, 16(5):289–304, June 2000.

[72] Seah H S and Lu J. Computer-assisted inbetweening of line drawings: image matching. In *CAD/Graphics*, 2001.

[73] Seah H S, Wu Z, Tian F, Xiao X, and Xie B. Artistic brushstroke representation and animation with disk B-spline curve. In *ACM SIGCHI International Conference on Advances in Computer Entertainment Technology (ACE 2005)*, pages 88–93, 2005.

[74] Su S, Xu Y, Shum H, and Chen F. Simulating artistic brushstrokes using interval splines. In *Proc. of the 5th International Conference on Computer Graphics and Imaging (CGIM)*, pages 85–90, 2002.

[75] K. Siddiqi, A. Shokoufandeh, S. J. Dickinson, and S. W. Zucker. Shock Graphs and Shape Matching. *International Journal of Computer Vision*, 35(1):13–32, 1999.

[76] Stern and Garland. GAS - a system for computer aided keyframe animation. *PhD dissertation, Univ. of Utah*, 1978.

[77] Beier T and Neely S. Feature-based image metamorphosis. *ACM/SIGGRAPH Computer Graphics*, (26(2)):35–42, 1992.

[78] Igarashi T, Matsuoka S, and Tanaka H. Teddy: a sketching interface for 3D freeform design. In *Proceedings of SIGGRAPH 1999*, pages 409–416, 1999.

[79] Reeves W T. Inbetweening for computer animation utilizing moving point constraints. In *Computer Graphics*, volume 15(3), pages 263–269, 1981.

[80] Surazhsky T, Surazhsky V, Barequet G, and Tal A. Blending polygonal shapes with different topologies. *Computers & Graphics*, 25(1):29–39, 2001.

[81] Kochanek D H U and Bartel R H. Interpolation splines with local tension, continuity, and bias control. In *Proceedings of ACM SIGGRAPH*, pages 33–41, 1984.

[82] Montanari U. A note on the minimal length polygonal approximation to a digitized contour. In *Communications of the ACM*, 13(1), pages 41–47, 1970.

[83] Patterson J W and Willis P J. Computer assisted animation: 2D or not 2D? *The Computer Journal*, (37(10)):829–839, 1994.

[84] Sederberg T W and Greenwood E. A physically based approach to 2-D shape blending. In *Computer Graphics, ACM*, pages 25–34, 1992.

[85] Sederberg T W and Greenwood E. Shape blending of 2-D piecewise curves. In *Mathematical methods for curves and surfaces*, Vanderbilt University Press, pages 497–506, 1999.

[86] Sederberg T W, Gao P, Wang G, and Mu H. 2-D shape blending: an intrinsic solution to the vertex path problem. In *Computer Graphics Proceedings*, Annual Conference Series, pages 15–18, 1993.

[87] Sederberg T W and Parry S R. Comparison of three curve intersection algorithms. In *Computer Aided Design*, volume 18(1), pages 58–63, 1986.

[88] Durand C X. The TOON project: requirement for a computerized 2D animation system. In *Proceedings of ACM SIGGRAPH*, pages 285–295, 1991.

[89] Hilaire X and Tombre K. Improving the accuracy of skeleton-based vectorization. In *Proceedings of the 4th IAPR International Workshop on Graphics Recognition, Kingston (Ontario, Canada)*, pages 381–394, September 2001.

[90] Xiao X, Seah H S, Wu Z, Tian F, and Xie X. Interactive free-hand drawing and in-between generation with disk B-spline curves. In *The Conference of Multimedia Arts Asia Pacific*, 2004.

[91] Guo Z and Hall R W. Parallel thinning with two-subiteration algorithms. In *Comm. ACM*, volume 32, pages 359–374, 1989.

[92] Wu Z, Seah H S, Tian F, Xiao X, and Xie X. Simulating artistic brushstrokes using disk B-spline curves. In *The Conference of Multimedia Arts Asia Pacific*, 2004.

[93] Zeng Z and Yan H. Region matching and optimal matching pair theorem. In *Computer Graphics International 2001*, 3-6, pages 232–239, July.

Publication

1. Qiu J, Seah HS, Tian F, Chen Q, Melikhov K. Computer-assisted auto coloring by region matching. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, 2003; pp. 175-184.

2. Chen Q, Tian F, Seah HS, Qiu J, Melikhov K. Automatic inbetweening based on feature matching. In *Proceedings of International Workshop on Advanced Image Technology*, 2004; pp. 347-351.

4. Chen Q, Tian F, Seah HS, Qiu J, Melikhov K. Initial value specification in motion estimation for keyframe animation. In *Proceedings of the Seventh IASTED International Conference on Computer Graphics and Imaging*, 2004; pp. 33-38.

3. Qiu J, Seah HS, Tian F, Wu Z, Chen Q. Topology enhanced feature-based auto coloring for computer-assisted cel animation. In *Proceedings of the Seventh IASTED International Conference on Computer Graphics and Imaging*, 2004; pp. 26-32.

5. Melikhov K, Tian F, Seah HS, Chen Q, Qiu J. Frame skeleton based auto-inbetweening in computer-assisted cel animation. In *Proceedings of the Third International Conference on Cyberworlds, 2004*; pp. 216-223.

6. Chen Q, Wu Z, Tian F, Seah H S, Qiu J, Melikhov K. Vectorization of raster line-drawings in cartoons. In *Proceedings of International Conference on Computer Animation and Social Agents, CASA*, pp. 151-158, Hong Kong, Oct 2005.

7. Qiu J, Seah HS, Tian F, Wu Z, Chen Q. Feature- and region-based auto painting for 2D animation. *The Visual Computer*, 21:928-944, Oct 2005

8. Qiu J, Seah HS, Tian F, Chen Q, Wu Z. Enhanced auto coloring with hierarchical region matching. *Computer Animation and Virtual Worlds*, 16(3-4):463-473, 2005;

10. Chen Q, Tian F, Seah HS, Qiu J, and Melikhov K. Motion estimation based on segmentation for keyframe inbetweening. In *Proceedings of International Workshop on Advanced Imaging Technology*, pp. 12-17, Jan 2006.

9. Qiu J, Seah HS, Tian F, Chen Q, and Melikhov K. Topology enhanced component segmentation for 2D animation character. In *Proceedings of International Workshop on Advanced Imaging Technology*, pp. 30-35, Jan 2006.

11. Melikhov K, Tian F, Qiu J, Chen Q and Seah HS. Grayscale line image vectorization and rendering. In *Proceedings of International Workshop on Advanced Imaging Technology*, pp. 591-596, Jan 2006.

12. Melikhov K, Tian F, Qiu J, Chen Q and Seah HS. DBSC-based grayscale line image vectorization. *Journal of Computer Science and Technology*, 21(2):244-248, Mar 2006.

13. Chen Q, Tian F, Seah HS, Wu Z, Qiu J and Melikhov K. DBSC-based animation enhanced with feature and motion. *Computer Animation and Virtual Worlds*, 17(3-4):189-198, 2006.

14. Qiu J, Seah HS, Tian F, Chen Q, Wu Z, and Melikhov K, Auto coloring with character registration, in *Proceedings of Joint International Conference on CyberGames and Interactive Entertainment 2006*, December 2006.